МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЕВА»
(Самарский университет)

В.В. ЗАЙЦЕВ

ЧИСЛЕННЫЕ МЕТОДЫ ДЛЯ ФИЗИКОВ. СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ И МАТРИЧНЫЕ ВЫЧИСЛЕНИЯ

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» в качестве учебного пособия для студентов, обучающихся по основной образовательной программе высшего образования по направлению подготовки 03.03.02 Физика

Самара
Издательство Самарского университета
2017

УДК 519.6(075)
ББК 22.19
З-17

Зайцев, Валерий Васильевич

З-17 **Численные методы для физиков. Системы линейных уравнений и матричные вычисления:** учеб. пособие / *В.В. Зайцев.* – Самара: Изд-во Самарского университета, 2017. – 72 с.

ISBN 978-5-7883-1353-5

Учебное пособие посвящено одному из разделов курса численных методов – системам линейных алгебраических уравнений и матричной проблеме собственных. Изложение проведено на «физическом» уровне строгости. Основное внимание уделено описанию численных алгоритмов и ограничениям и проблемам, возникающим при их применении.

Приведены примеры реализации численных алгоритмов с использованием пакета Mathcad.

Предназначено для студентов изучающих курс «Численные методы и математическое моделирование».

УДК 519.6(075)
ББК 22.19

ISBN 978-5-7883-1353-5

ПРЕДИСЛОВИЕ

Учебное пособие написано на основе лекций, читаемых автором студентам-физикам Самарского университета, и посвящено одному из разделов курса численных методов – системам линейных алгебраических уравнений и матричной проблеме собственных.

В пособии дано описание основных численных методов решения СЛАУ – прямых и итерационных, а также методов решения задач на собственные значения матриц. Рассмотрены как традиционные, классические численные методы, так и методы, вошедшие в вычислительную практику сравнительно недавно.

Изложение проводится на «физическом» уровне строгости. Математические обоснования большинства методов даны на основе элементарных результатов математического анализа и должны быть понятны студентам младших курсов. Основное внимание уделено практической стороне использования численных методов, а также ограничениям и проблемам, возникающим при их применении. Там, где это возможно, описание математических результатов сопровождается физической интерпретацией процессов, лежащих в основе численных алгоритмов.

Приведены многочисленные примеры реализации рассмотренных численных алгоритмов с использованием пакета Mathcad. Предполагается, что студенты прослушали курс программирования и имеют навыки составления программ. Детальный разбор программных модулей Mathcad, имеющих «прозрачную» структуру, часто способных заменить стандартные блок-схемы алгоритмов, рекомендуется для наиболее полного понимания сути изучаемых методов. Кроме того, использование Mathcad дает возможность студентам совершенствоваться в научном программировании.

Печатается по решению учебно-методического совета естественнонаучного института.

ГЛАВА 1. ПРЯМЫЕ МЕТОДЫ РЕШЕНИЯ СЛАУ

1.1. Системы линейных алгебраических уравнений и методы их решения

Стандартная развёрнутая запись системы линейных алгебраических уравнений (СЛАУ) имеет вид

$$\begin{aligned} &a_{11}x_1 + a_{12}x_2 + ... + a_{1N}x_N = b_1, \\ &a_{21}x_1 + a_{22}x_2 + ... + a_{2N}x_N = b_2, \\ &... \\ &a_{N1}x_1 + a_{N2}x_2 + ... + a_{NN}x_N = b_N. \end{aligned} \qquad (1.1)$$

Здесь коэффициенты системы a_{nm} (элементы квадратной матрицы **A**) и правые части системы b_n (элементы вектора правых частей **b**) являются известными величинами, а величины x_n, составляющие вектор решения **x**, подлежат определению. Отметим, что в записи двойного индекса коэффициентов a_{nm} подразумевается запятая, так что полная запись выглядит как $a_{n,m}$. Но в дальнейшем, если это не приводит к неоднозначности, запятая пропускается.

В компактной форме СЛАУ (1.1) записывается как

$$\sum_{m=1}^{N} a_{nm}x_m = b_n, \; n = 1,2,...,N, \qquad (1.1')$$

а в векторно-матричной как

$$\mathbf{Ax} = \mathbf{b}. \qquad (1.1'')$$

СЛАУ возникают непосредственным образом при решении многих физических задач: например, при расчёте электрических цепей на основе законов Кирхгофа. Кроме того, к решению СЛАУ сводится расчёт методом конечных разностей физических процессов и полей, описываемых дифференциальными уравнениями. И, наконец, часто решение СЛАУ является промежуточным этапом вычислений по некоторому алгоритму. Например, алгоритм поиска корней системы нелинейных уравнений методом Ньютона включает в себя решение системы линейных уравнений с матрицей Якоби.

В теоретических курсах линейной алгебры в качестве классического метода решения СЛАУ рассматривается метод Крамера. В его основе лежит равенство

$$\left(c_1x_1+c_2x_2+...+c_Nx_N\right)Det\left(\mathbf{A}\right)=-Det\begin{pmatrix} a_{11} & a_{12} & ... & a_{1N} & b_1 \\ a_{21} & a_{22} & ... & a_{2N} & b_2 \\ ... & ... & ... & ... & ... \\ a_{N1} & a_{N2} & ... & a_{NN} & b_N \\ c_1 & c_2 & ... & c_N & 0 \end{pmatrix}, \qquad (1.2)$$

справедливое для любых чисел c_n. Если в (1.2) положить $c_n=0$ для всех $n\neq m$ и $c_m=1$, то нетрудно получить

$$x_m=(-1)^{N+m+1}\left[Det\left(\mathbf{A}\right)\right]^{-1}Det\begin{pmatrix} ... & a_{1,m-1} & a_{1,m+1} & ... & b_1 \\ ... & a_{2,m-1} & a_{2,m+1} & ... & b_2 \\ ... & ... & ... & ... & ... \\ ... & a_{N-1,m-1} & a_{N-1,m+1} & ... & b_{N-1} \\ ... & a_{N,m-1} & a_{N,m+1} & ... & b_N \end{pmatrix}. \qquad (1.3)$$

Формула (1.3) проста, хотя её запись и довольно громоздка. Но при практических вычислениях она чрезвычайно неэффективна. Действительно, для вычисления определителя матрицы размером $N\times N$ требуется $N!$ операций умножения. Уже при $N=20$, а для метода конечных элементов это совсем небольшая матрица, $N!\sim 2.4\cdot 10^{18}$. Пусть операция умножения выполняется на компьютере за $10^{-7}c$, тогда на вычисление определителя такой матрицы потребуется примерно 7715 *лет*. На порядок более быстрый компьютер проведёт вычисления за 771 *год*. Вы согласны ждать это время?

Применяемые на практике методы численного решения СЛАУ делятся на две группы: прямые и итерационные.

Прямые методы позволяют получить точное решение $\mathbf{x}_r$ задачи (1.1) за конечное число арифметических операций, но при условии, что операции выполняются без арифметических погрешностей. При наличии погрешностей в арифметических операциях решение, естественно, получается приближённым.

Итерационные методы позволяют вычислять последовательность приближений $\left\{\mathbf{x}^{(k)}\right\}$, сходящуюся к решению $\mathbf{x}_r$ при $k\to\infty$. На практике последовательность содержит конечное число приближений в зависимости от заданного значения нормы вектора ошибки $\left\|\mathbf{x}^{(k)}-\mathbf{x}_r\right\|$.

1.2. Метод исключения Гаусса

Алгоритм метода делится на два этапа: прямой ход, приводящий исходную систему (1.1) к системе с верхней треугольной матрицей, и обратный ход, заключающийся в решении СЛАУ с треугольной матрицей.

Прямой ход. Будем считать, что $a_{11} \neq 0$. Иначе можно переставить уравнения в системе (1.1) так, чтобы это условие выполнялось. Для унификации обозначений в коэффициенты первого уравнения добавим верхний индекс: $a_{11}^{(1)} = a_{11},\ a_{12}^{(1)} = a_{11},...,\ a_{1N}^{(1)} = a_{1N},\ \ b_1^{(1)} = b_1$.

Преобразуем исходную СЛАУ так, чтобы в преобразованной системе во всех уравнениях, начиная со второго, компонента вектора решения x_1 отсутствовала. Для этого будем последовательно умножать первое уравнение на коэффициент

$$l_{n1} = \frac{a_{n1}^{(1)}}{a_{11}^{(1)}},\ n = 2,3,...,N$$

и вычитать результат умножения из *n*-ного уравнения. После всех умножений и вычитаний получим СЛАУ следующего вида

$$\begin{aligned} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + a_{13}^{(1)}x_3 + ... + a_{1N}^{(1)}x_N &= b_1^{(1)}, \\ a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 + ... + a_{2N}^{(2)}x_N &= b_2^{(2)}, \\ ... & \\ a_{N2}^{(2)}x_2 + a_{N3}^{(2)}x_3 + ... + a_{NN}^{(2)}x_N &= b_N^{(2)}, \end{aligned} \qquad (1.4)$$

где отличные от нуля элементы видоизменённых частей матрицы **A** и вектора **b** равны

$$a_{nm}^{(2)} = a_{nm}^{(1)} - l_{n1}a_{1m}^{(1)},\ b_n^{(2)} = b_n - l_{n1}b_1^{(1)};\ \ n = 2,3,...,N,\ \ m = 2,3,...,N.$$

Теперь аналогичным преобразованиям с целью исключения компоненты x_2 подвергнем подсистему системы (1.4), состоящую из $(N-1)$-го уравнения, начиная со второго. В результате преобразований получим подсистему

$$\begin{aligned} a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 + ... + a_{2N}^{(2)}x_N &= b_2^{(2)}, \\ a_{33}^{(3)}x_3 + ... + a_{3N}^{(3)}x_N &= b_3^{(3)}, \\ .... & \\ a_{N3}^{(3)}x_3 + ... + a_{NN}^{(3)}x_N &= b_N^{(3)} \end{aligned} \qquad (1.5)$$

с изменёнными элементами матрицы **A** и вектора **b**:

$$l_{n2}=\frac{a_{n2}^{(2)}}{a_{22}^{(2)}},\ a_{nm}^{(3)}=a_{nm}^{(2)}-l_{n2}a_{2m}^{(2)},\ b_n^{(3)}=b_n^{(2)}-l_{n2}b_2^{(2)};$$

$$n=3,4,...,N,\ m=3,4,...,N.$$

Продолжая эти преобразования СЛАУ, будем исключать на каждом шаге последовательно компоненты x_3, x_4 и так далее. Нетрудно записать формулы преобразования элементов матрицы СЛАУ и вектора правых частей на *k*-ом шаге исключения:

$$l_{nk}=\frac{a_{nk}^{(k)}}{a_{kk}^{(k)}},\ a_{nm}^{(k+1)}=a_{nm}^{(k)}-l_{nk}a_{km}^{(k)},\ b_n^{(k+1)}=b_n^{(k)}-l_{nk}b_k^{(k)};$$

$$n=k+1,k+2,...,N,\ m=k=1,k+2,...,N. \tag{1.6}$$

В конечном результате процесс исключения переменных приводит к системе уравнений с верхней треугольной матрицей, чем и завершается этап прямого хода:

$$\begin{aligned}
a_{11}^{(1)}x_1+a_{12}^{(1)}x_2+a_{13}^{(1)}x_3+...+a_{1N}^{(1)}x_N&=b_1^{(1)},\\
a_{22}^{(2)}x_2+a_{23}^{(2)}x_3+...+a_{2N}^{(2)}x_N&=b_2^{(2)},\\
a_{33}^{(3)}x_3+...+a_{3N}^{(3)}x_N&=b_3^{(3)},\\
&...\\
a_{NN}^{(N)}x_N&=b_N^{(N)}.
\end{aligned} \tag{1.7}$$

При этом константы последнего шага исключения имеют вид

$$l_{N,N-1}=\frac{a_{N,N-1}^{(N-1)}}{a_{N-1,N-1}^{(N-1)}},\ a_{NN}^{(N)}=a_{NN}^{(N-1)}-l_{N,N-1}a_{N-1,N}^{(N-1)},\ b_N^{(N)}=b_N^{(N-1)}-l_{N,N-1}b_{N-1}^{(N-1)}.$$

Матрицу системы (1.7) принято обозначать через **U**:

$$\mathbf{U}=\begin{pmatrix}
u_{11} & u_{12} & u_{13} & ... & u_{1N}\\
0 & u_{22} & u_{23} & ... & u_{2N}\\
0 & 0 & u_{33} & ... & u_{3N}\\
... & ... & ... & ... & ...\\
0 & 0 & 0 & ... & u_{NN}
\end{pmatrix}=\begin{pmatrix}
a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & ... & a_{1N}^{(1)}\\
0 & a_{22}^{(2)} & a_{23}^{(2)} & ... & a_{2N}^{(2)}\\
0 & 0 & a_{33}^{(3)} & ... & a_{3N}^{(3)}\\
... & ... & ... & ... & ...\\
0 & 0 & 0 & ... & a_{NN}^{(N)}
\end{pmatrix}.$$

Mathcad-функция *FS(A,b)* реализует вычисления прямого хода по формулам (1.6).

```
FS(A,b) := | Прямой_ход_метода_Гаусса← %
          | без_выделения_ведущего_элемента ← %
          | N ← rows(A)
          | A ← augment(A,b)
          | for k ∈ 0.. N − 2
          |   for i∈ k + 1.. N − 1
          |     for j ∈ k + 1.. N
          |       L_{i,k} ← A_{i,k} / A_{k,k}
          |       A_{i,j} ← A_{i,j} − L_{i,k}·A_{k,j}
          | for i∈ 0.. N − 1
          |   for j ∈ i.. N − 1
          |     U_{i,j} ← A_{i,j}
          | augment(U, A^⟨N⟩)
```

Оценка количества вычислений на этапе прямого хода даёт величину $N_{FS} \sim 2N^3/3$, которая при больших N несравнимо меньше, чем $N!$ вычислений по методу Крамера.

Обратный ход. Решение СЛАУ с верхней треугольной матрицей

$$\mathbf{Ux} = \mathbf{y},$$

где вектор правых частей $\mathbf{y}^T = \left(b_1^{(1)}, b_2^{(2)}, ..., b_N^{(N)}\right)$, не представляет затруднений и проводится по рекуррентной формуле

$$x_N = \frac{1}{u_{NN}} y_N,\ x_n = \frac{1}{u_{nn}}\left(y_n - \sum_{m=n+1}^{N} u_{nm} x_m\right),\ n = N-1, N-2, ..., 1. \qquad (1.8)$$

Функция *BS(A,b)* – пример реализации обратного хода в Mathcad.

Подсчёт количества вычислений по алгоритму обратного хода (1.8) даёт величину $N_{BS} = N^2 << N_{FS}$. Поэтому основные вычислительные затраты при решении СЛАУ методом Гаусса приходятся на прямой ход метода.

$$
\begin{aligned}
&\mathrm{BS}(U,b) := \left|
\begin{array}{l}
\text{Обратный\_ход\_метода\_Гаусса} \leftarrow \% \\
N \leftarrow \mathrm{rows}(U) - 1 \\
x_N \leftarrow \dfrac{1}{U_{N,N}} \cdot b_N \\
\text{for } n \in N-1\,..\,0 \\
\quad x_n \leftarrow \dfrac{1}{U_{n,n}} \cdot \left(b_n - \sum_{m=n+1}^{N} U_{n,m} \cdot x_m \right) \\
x
\end{array}
\right.
\end{aligned}
$$

С помощью функции *LSOLVE(A,b)*, организованной по модульному принципу, можно решить СЛАУ (1.1) в пакете Mathcad.

$$
\begin{aligned}
&\mathrm{LSOLVE}(A,b) := \left|
\begin{array}{l}
N \leftarrow \mathrm{rows}(A) \\
Ub \leftarrow \mathrm{FS}(A,b) \\
U \leftarrow \mathrm{submatrix}(Ub,0,N-1,0,N-1) \\
b \leftarrow Ub^{\langle N \rangle} \\
\mathrm{BS}(U,b)
\end{array}
\right.
\end{aligned}
$$

Пример решения системы из трёх уравнений даёт программа (Pr.1). В ней, помимо функции *LSOLVE(A,b)*, для сопоставления результатов также использована встроенная Mathcad-функция *lsolve(A,b)*.

$$
A = \begin{pmatrix} 2 & 3 & 5 \\ 7 & 2 & 9 \\ 6 & 19 & 4 \end{pmatrix}
$$

$$
b^{T} = (\,1 \quad 2 \quad 3\,) \qquad \mathrm{lsolve}(A,b)^{T} = (\,0.151 \quad 0.092 \quad 0.084\,)
$$

$$
\mathrm{LSOLVE}(A,b)^{T} = (\,0.151 \quad 0.092 \quad 0.084\,)
$$

(Pr.1)

Выбор главного элемента. Вновь обратимся к этапу прямого хода. Понятно, что формулы вида (1.6) имеют смысл только тогда, когда все элементы $a_{kk}^{(k)} \neq 0$. Эти элементы матриц называются главными (ведущими). Если в первом уравнении (1.1) $a_{11}^{(1)}$ можно гарантированно сделать отличным

от нуля «ручной» перестановкой уравнений, то на всех последующих шагах исключения выбор главного элемента с перестановкой уравнений приходится осуществлять программно. Например, это позволяет сделать Mathcad-функция *GE(A,k)*.

```
GE(A,k) ≡ | Выбор_ведущего_элемента ← %
          | (путем_перестановки_строк ← %)
          | N ← rows(A)
          | B ← submatrix(A,k,N − 1,k,N − 1)
          | Bmax ← B0,0
          | for i ∈ 0.. N − k − 1
          |   if |Bi,0| ≥ Bmax
          |     | Bmax ← |Bi,0|
          |     | M ← i
          | for j ∈ 0.. N − 0
          |     | c ← Ak,j
          |     | Ak,j ← AM+k,j
          |     | AM+k,j ← c
          | A
```

Функция *GE(A,k)* в вычислениях прямого хода используется так, как это сделано в *FSG(A,b)*.

Выбор путём перестановки строк матрицы СЛАУ не является единственным способом выбора главного элемента. Применяется также способ с перестановкой столбцов. Кроме того, полный выбор ведущего элемента состоит в перестановке как строк, так и столбцов.

Отметим, что метод исключения с выбором главного элемента, помимо устранения проблемы с возможным делением на ноль, устраняет также и деление на малые числа, ведущее к потере точности вычислений.

```
FSG(A,b) := | Прямой_ход_метода_Гаусса← %
            | с_выделением_ведущего_элемента ← %
            | N ← rows(A)
            | A ← augment(A,b)
            | for k ∈ 0.. N − 2
            |     A ← GE(A,k)
            |     for i ∈ k + 1.. N − 1
            |       for j ∈ k + 1.. N
            |           L_{i,k} ← A_{i,k} / A_{k,k}
            |           A_{i,j} ← A_{i,j} − L_{i,k}·A_{k,j}
            | for i ∈ 0.. N − 1
            |   for j ∈ i.. N − 1
            |     U_{i,j} ← A_{i,j}
            | augment(U, A^⟨N⟩)
```

1.3. Исключение Гаусса и *LU*-разложение

После выполнения этапа прямого хода метода исключения Гаусса СЛАУ (1.1″) приводится к виду (1.7)

$$\mathbf{Ux} = \mathbf{y}, \tag{1.9}$$

или в развёрнутой форме

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1N} \\ 0 & u_{22} & u_{23} & \dots & u_{2N} \\ 0 & 0 & u_{33} & \dots & u_{3N} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & u_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_N \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_N \end{pmatrix}.$$

Элементы верхней треугольной матрицы **U,** как было установлено, определяются рекуррентными формулами (см. (1.6))

$$u_{1m} = a_{1m}^{(1)},\ a_{nm}^{(1)} = a_{nm},\ n = 1,2,...,N,\ m = 1,2,...,N;$$

$$u_{2m} = a_{2m}^{(2)},\ a_{nm}^{(2)} = a_{nm}^{(1)} - \frac{a_{n1}^{(1)}}{a_{11}^{(1)}} a_{1m}^{(1)},\ n = 2,3,...,N,\ m = 2,3,...,N;$$

$$u_{3m} = a_{3m}^{(3)},\ a_{nm}^{(3)} = a_{nm}^{(2)} - \frac{a_{n2}^{(2)}}{a_{22}^{(2)}} a_{2m}^{(2)},\ n = 3,4,...,N,\ m = 3,4,...,N;\ ...$$

$$u_{N,N} = a_{N,N}^{(N)},\ a_{N,N}^{(N)} = a_{N,N}^{(N-1)} - \frac{a_{N,N-1}^{(N-1)}}{a_{N-1,N-1}^{(N-1)}} a_{N-1,N}^{(N-1)}.$$

Что же касается вектора $\mathbf{y}^T = \left(b_1^{(1)}, b_2^{(2)}, ..., b_N^{(N)}\right)$, то он связан с правыми частями исходной системы уравнений следующим образом:

$$\begin{aligned}
&y_1 = b_1,\\
&y_2 = b_2 - l_{21} y_1,\\
&y_3 = b_3 - l_{31} y_1 - l_{32} y_2,\\
&...\\
&y_N = b_N - l_{N,1} y_1 - ... - l_{N,N-2} y_{N-2} - l_{N,N-1} y_{N-1}.
\end{aligned}$$

Нетрудно заметить, что совокупность этих соотношений представляет собой СЛАУ

$$\mathbf{Ly} = \mathbf{b} \tag{1.10}$$

с нижней треугольной матрицей, составленной из множителей l_{nm} прямого хода:

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & ... & 0 \\ l_{21} & 1 & 0 & ... & 0 \\ l_{31} & l_{32} & 1 & ... & 0 \\ ... & ... & ... & ... & ... \\ l_{N1} & l_{N2} & l_{N3} & ... & 1 \end{pmatrix},$$

где

$$l_{nm} = \frac{a_{nm}^{(m)}}{a_{mm}^{(m)}},\ n = 1,2,...,N,\ m = 1,2,...,n.$$

Умножив систему (1.9) слева на матрицу **L**, с учётом системы (1.10) получим

$$\mathbf{LUx} = \mathbf{Ly} = \mathbf{b} = \mathbf{Ax}.$$

Эти равенства выполняются при любом выборе вектора правых частей **b** и, следовательно, при любом векторе решения **x**. Значит, матрицы **L** и **U** можно рассматривать как сомножители матрицы **A**:

$$\mathbf{A} = \mathbf{LU}. \tag{1.11}$$

Такое разложение квадратной матрицы в произведение нижней и верхней треугольных матриц называется *LU*-разложением матрицы **A**.

Разложение (1.11) имеет место, если на этапе прямого хода не производились перестановки уравнений, связанные с выбором главных элементов. В том случае, когда перестановки имели место, *LU*-разложение применяется к произведению матрицы перестановок **P** с исходной матрицей **A** и выглядит как

$$\mathbf{PA} = \mathbf{LU}. \tag{1.12}$$

Матрица **P** формируется из единичной матрицы путём таких же перестановок её строк, что и перестановки уравнений при выборе главных элементов.

Функция *LU(A)* даёт пример реализации разложения в Mathcad. Пример применения – программа (Pr.2), из которого, в частности, следует, что матрица перестановок ортогональна. Имя встроенной функции Mathcad для *LU*-разложения – *lu(A)*.

Отметим, что *LU*-разложение является лишь формой представления алгоритма исключения Гаусса. Тем не менее, часто оно весьма полезно в матричных вычислениях.

$$B = \begin{pmatrix} 2 & 3 & 5 \\ 7 & 2 & 9 \\ 6 & 19 & 2 \end{pmatrix} \qquad P := \text{submatrix}(LU(B), 0, 2, 0, 2)$$

$$LU(B) = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 7 & 2 & 9 \\ 0 & 0 & 1 & 0.857 & 1 & 0 & 0 & 17.286 & -5.714 \\ 1 & 0 & 0 & 0.286 & 0.14 & 1 & 0 & 0 & 3.231 \end{pmatrix}$$

$$lu(B) = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 7 & 2 & 9 \\ 0 & 0 & 1 & 0.857 & 1 & 0 & 0 & 17.286 & -5.714 \\ 1 & 0 & 0 & 0.286 & 0.14 & 1 & 0 & 0 & 3.231 \end{pmatrix}$$

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \quad P^{T} \cdot P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad P \cdot P^{T} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{Pr.2}$$

```
LU(A) := | LU_разложение_матрицы ← %
         | с_перестановкой_строк ← %
         | N ← rows(A)
         | I ← identity(N)
         | A ← augment(A,I)
         | for k ∈ 0.. N – 2
         |     A ← GE(A,N,k)
         |     for i ∈ k + 1.. N – 1
         |         L_{i,k} ← A_{i,k} / A_{k,k}
         |         for j ∈ k + 1.. N – 1
         |             A_{i,j} ← A_{i,j} – L_{i,k}·A_{k,j}
         |         A_{i,k} ← L_{i,k}
         | for i ∈ 0.. N – 1
         |     for j ∈ i.. N – 1
         |         U_{i,j} ← A_{i,j}
         | L ← submatrix(A,0,N – 1,0,N – 1) – U + I
         | P ← submatrix(A,0,N – 1,N,2·N – 1)
         | augment(L,U,P)
```

1.4. Вычисление обратной матрицы и определителя

По определению обратной матрицы $\mathbf{A}^{-1}$ справедливо матричное равенство

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}.$$

Представим единичную матрицу как объединение векторов $\mathbf{e}_n$, у каждого из которых отлична от нуля и равна единице лишь *n*-ая составляющая, то есть

$$\mathbf{I} = (\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_N).$$

Тогда *n*-ый столбец обратной матрицы будет вектором $\mathbf{X}_n$ решения уравнения

$$\mathbf{A}\mathbf{X}_n = \mathbf{e}_n \ . \qquad (1.13)$$

При этом для решения N уравнений вида (1.13) достаточно провести одно LU-разложение (1.12) и определять столбцы $\mathbf{X}_n$ прямыми и обратными подстановками:

$$\mathbf{LY}_n = \mathbf{Pe}_n, \;\; \mathbf{UX}_n = \mathbf{Y}_n.$$

Примером вычисления обратной матрицы в Mathcad служит функция *INV(A)*.

```
INV(A) := | Обращение_матрицы ← %
          | на_основе_LU_разложения ← %
          | N ← rows(A)
          | PLU ← LU(A)
          | P ← submatrix(PLU,0,N − 1,0,N − 1)
          | L ← submatrix(PLU,0,N − 1,N,2·N − 1)
          | U ← submatrix(PLU,0,N − 1,2·N,3·N − 1)
          | B ← P·identity(N)
          | for m ∈ 0.. N − 1
          |   | b ← B^⟨m⟩
          |   | Y_0 ← b_0
          |   | for n ∈ 1.. N − 1
          |   |   Y_n ← b_n − Σ_{m=0}^{n−1} L_{n,m}·Y_m
          |   | Ainv^⟨m⟩ ← BS(U,Y)
          | Ainv
```

Для вычисления определителя матрицы можно использовать её LU-разложение (1.12) и равенства

$$Det(\mathbf{PA}) = Det(\mathbf{P})Det(\mathbf{A}) = Det(\mathbf{L})Det(\mathbf{U}) = Det(\mathbf{LU}),$$

из которых следует

$$Det(\mathbf{A}) = Det(\mathbf{P})^{-1} Det(\mathbf{L})Det(\mathbf{U}). \qquad (1.14)$$

С учётом того, что каждая перестановка строк матрицы меняет знак её определителя и $Det(\mathbf{I}) = 1$, имеем

$$Det(\mathbf{P}) = (-1)^{\nu},$$

где ν – число перестановок в процессе *LU*-разложения. Функция *NP(P)* производит подсчёт перестановок.

```
NP(P) ≡ | Число_перестановок_в_P ← %
        | N ← rows(P)
        | for n ∈ 0.. N – 1
        |   for m ∈ 0.. N – 1
        |     K_n ← m  if P_n,m = 1
        | ν ← 0
        | for n ∈ 0.. N – 1
        |   if K_n ≠ n
        |     | K_(K_n) ← K_n
        |     | K_n ← n
        |     | ν ← ν + 1
        | ν
```

Нетрудно также установить, что определитель треугольной матрицы равен произведению её диагональных элементов. Поэтому выражение (1.14) сводится к расчётной формуле

$$Det(\mathbf{A}) = (-1)^{\nu} U_{11} U_{22} ... U_{NN} .$$

Эти вычисления реализует функция *Det(A)*.

```
Det(A) := | Вычисление_определителя_матрицы ← %
          | N ← rows(A)
          | PLU ← LU(A)
          | P ← submatrix(PLU,0,N – 1,0,N – 1)
          | U ← submatrix(PLU,0,N – 1,2·N,3·N – 1)
          | ν ← NP(P)
          | D ← (–1)^ν · ∏_{n=0}^{N–1} U_n,n
```

1.5. Метод исключения Гаусса–Жордана

Алгоритм этого метода состоит в том, что матрица СЛАУ приводится к диагональной форме, а не треугольной, как в методе Гаусса. В процессе исключения преобразуются уравнения, стоящие как перед ведущим, так и после него. Но сама процедура исключения не меняется – это умножение и вычитание уравнений.

Mathcad-функция *GJ(A,b)* реализует метод исключения Гаусса–Жордана.

```
GJ(A,b) := | Решение_СЛАУ ← %
           | методом_исключения_Гаусса_Жордана ← %
           | N ← rows(A)
           | A ← augment(A,b)
           | for k ∈ 0..N − 2
           |   for i ∈ k + 1..N − 1
           |     | for j ∈ k + 1..N
           |     |   A[i,j] ← A[i,j] − A[i,k]·A[k,j]/A[k,k]
           |     | A[i,k] ← 0
           | for k ∈ 1..N − 1
           |   for i ∈ k − 1..0
           |     | for j ∈ k + 1..N
           |     |   A[i,j] ← A[i,j] − A[i,k]·A[k,j]/A[k,k]
           |     | A[i,k] ← 0
           | for i ∈ 0..N − 1
           |   x[i] ← A[i,N]/A[i,i]
           | x
```

$$A_{i,j} \leftarrow A_{i,j} - A_{i,k} \cdot \frac{A_{k,j}}{A_{k,k}}, \qquad x_i \leftarrow \frac{A_{i,N}}{A_{i,i}}$$

Ниже приведён пример преобразования СЛАУ с матрицей **A** и вектором правых частей **b** демонстрационной функцией *GJd(A,b)* (в отличие от основной она возвращает не вектор решения, а расширенную матрицу системы).

$$\mathrm{A}=\begin{pmatrix}2&3&5&7\\7&2&9&8\\6&19&2&10\\2&7&3&5\end{pmatrix}\quad \mathrm{b}=\begin{pmatrix}1\\2\\3\\4\end{pmatrix}\quad \mathrm{GJd}(\mathrm{A},\mathrm{b})=\begin{pmatrix}2&0&0&0&-1.891\\0&-8.5&0&0&-8.761\\0&0&-23&0&-47.51\\0&0&0&-1.831&2.754\end{pmatrix}$$

Очевидно, в методе Гаусса–Жордана обратный ход фактически отсутствует – компоненты вектора решения вычисляются через элементы преобразованных матрицы $\mathbf{A}'$ и вектора $\mathbf{b}'$ по формуле

$$x_n=\frac{b'_n}{A'_{nn}}.$$

Однако этап прямого хода требует большего количества вычислений, чем в методе Гаусса.

1.6. Положительно определённые системы и разложение Холесского

В физических приложениях часто встречаются СЛАУ, матрицы коэффициентов которых имеют специальный вид, а именно являются симметричными и положительно определёнными. Сокращённое обозначение таких матриц – *СПО*-матрицы (*SPD*-матрицы). Для них справедливо так называемое разложение Холесского

$$\mathbf{A}=\mathbf{R}^T\mathbf{R}, \tag{1.15}$$

где $\mathbf{R}$ – верхняя треугольная матрица с положительными элементами на главной диагонали.

С использованием разложения (1.14) СЛАУ $\mathbf{Ax}=\mathbf{b}$ распадается на две подсистемы:

$$\mathbf{R}^T\mathbf{y}=\mathbf{b},\ \mathbf{Rx}=\mathbf{y}. \tag{1.16}$$

Каждая из них имеет треугольную матрицу и решается подстановкой (прямой или обратной).

Поэлементный анализ разложения (1.15) позволяет определить его алгоритм. Действительно, нетрудно установить, что элементы первой строки матрицы $\mathbf{A}$ связаны с элементами r_{nm} матрицы $\mathbf{R}$ соотношениями

$$a_{1m}=r_{11}r_{1m},\ m=1,2,...,N.$$

Следовательно,

$$r_{11}=\sqrt{a_{11}},\ r_{1m}=a_{1m}r_{11}^{-1},\ m=2,3,...,N. \tag{1.17}$$

Для второй строки матрицы $\mathbf{A}$ можно записать соотношения

$$a_{2m}=r_{12}r_{1m}+r_{22}r_{2m},\ m=2,3,...,N.$$

Откуда следует

$$r_{22} = \sqrt{a_{22} - r_{12}^2},\ r_{2m} = \left(a_{2m} - r_{12} r_{1m}\right) r_{22}^{-1},\ m = 3,4,...,N. \qquad (1.18)$$

Продолжая этот анализ, найдём общие выражения вида

$$r_{nn} = \sqrt{a_{nn} - \sum_{k=1}^{n-1} r_{km}^2},\ r_{nm} = \left(a_{nm} - \sum_{k=1}^{n-1} r_{kn} r_{km} \right) r_{nn}^{-1},\ m = n+1, ..., N. \qquad (1.19)$$

Рекуррентные формулы (1.17) – (1.19) легко программируются и составляют основное содержание Mathcad-функции *CHOLES(A)*.

```
CHOLES(A) := | Разложение_Холесского ← %
             | N ← rows(A)
             | for n ∈ 0.. N – 1
             |   if n = 0
             |     A0,0 ← √A0,0
             |     for m ∈ 1.. N – 1
             |       Am,0 ← Am,0 / A0,0
             |   otherwise
             |     An,n ← √(An,n – Σ_{k=0}^{n–1} (Ak,n)^2)
             |     for m ∈ n.. N – 1
             |       if m ≠ n
             |         S ← Am,n – Σ_{k=0}^{n–1} Ak,n·Ak,m
             |         An,m ← (An,n)^–1·S
             |     for m ∈ 0.. n – 1
             |       An,m ← 0
             | A
```

Программа (Pr.3) демонстрирует пример разложения Холесского.

$$\text{AC} := \text{CHOLES}(\text{A}) \qquad \text{H} := \text{cholesky}(\text{A})$$

$$\text{AC} = \begin{pmatrix} 2 & -1 & 2 & 1 \\ 0 & 3 & 0 & -2 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \text{H} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ -1 & 3 & 0 & 0 \\ 2 & 0 & 2 & 0 \\ 1 & -2 & 1 & 1 \end{pmatrix}$$

$$\text{A} \equiv \begin{pmatrix} 4 & -2 & 4 & 2 \\ -2 & 10 & -2 & -7 \\ 4 & -2 & 8 & 4 \\ 2 & -7 & 4 & 7 \end{pmatrix} \qquad \text{AC}^{\text{T}} \cdot \text{AC} = \begin{pmatrix} 4 & -2 & 4 & 2 \\ -2 & 10 & -2 & -7 \\ 4 & -2 & 8 & 4 \\ 2 & -7 & 4 & 7 \end{pmatrix} \qquad \text{(Pr.3)}$$

Отметим, что из приведённого примера также следует, что встроенная функция пакета Mathcad возвращает нижнюю треугольную матрицу $\mathbf{L} = \mathbf{R}^T$.

Метод решения СЛАУ с использованием представления системы в виде (1.15) известен также как метод квадратных корней.

1.7. Метод прогонки для СЛАУ с трёхдиагональными матрицами

Трёхдиагональными называются матрицы, у которых отличны от нуля лишь элементы, расположенные на главной диагонали, а также на верхней и нижней поддиагоналях. СЛАУ с такими матрицами часто встречаются, например, при решении уравнений математической физики методами конечных разностей и в других физических приложениях.

СЛАУ с трёхдиагональной матрицей запишем в виде

$$\begin{aligned}
&d_1 x_1 + d_2^{(+)} x_2 = b_1, \\
&d_1^{(-)} x_1 + d_2 x_2 + d_3^{(+)} x_3 = b_2, \\
&d_{n-1}^{(-)} x_{n-1} + d_n x_n + d_{n+1}^{(+)} x_{n+1} = b_n, \ n = 3,4,...,N-2, \\
&d_{N-2}^{(+)} x_{N-1} + d_{N-1} x_{N-1} + d_N^{(+)} x_N = b_{N-1}, \\
&d_{N-1}^{(+)} x_{N-1} + d_N^{(+)} x_N = b_N .
\end{aligned} \qquad (1.20)$$

В векторно-матричной форме эта система выглядит как

$$\begin{pmatrix} d_1 & d_2^{(+)} & ... & ... & ... & 0 & 0 \\ d_1^{(-)} & d_2 & ... & ... & ... & 0 & 0 \\ 0 & d_2^{(-)} & ... & d_n^{(+)} & ... & 0 & 0 \\ ... & ... & ... & d_n & ... & ... & ... \\ 0 & 0 & ... & d_n^{(-)} & ... & d_{N-1}^{(+)} & 0 \\ 0 & 0 & ... & ... & ... & d_{N-1} & d_N^{(+)} \\ 0 & 0 & ... & ... & ... & d_{N-1}^{(-)} & d_N \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ ... \\ x_{N-2} \\ x_{N-1} \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ ... \\ b_{N-2} \\ b_{N-1} \\ b_N \end{pmatrix}. \qquad (1.20')$$

Первое из уравнений в (1.20) связывает компоненты решения x_1 и x_2 следующим образом

$$x_1 = -\frac{d_2^{(+)}}{d_1} x_2 + \frac{b_1}{d_1}. \qquad (1.21)$$

Пусть связь аналогичного вида имеет место и в общем случае:

$$x_k = \alpha_k x_{k+1} + \gamma_k,\ k = 1,2...,N-1, \qquad (1.22)$$

где α_k и γ_k – неизвестные коэффициенты. Для их определения подставим x_{n-1} из (1.22) в уравнение общего виды системы (1.20). После несложных преобразований получим

$$x_n = -\frac{d_{n+1}^{(+)}}{\alpha_{n-1} d_{n-1}^{(-)} + d_n} x_{n+1} + \frac{b_n - \gamma_{n-1} d_{n-1}^{(-)}}{\alpha_{n-1} d_{n-1}^{(-)} + d_n},\ n = 2,3,...,N-1. \qquad (1.23)$$

Сравнив соотношения (1.22) и (1.23), приходим к выводу о том, что коэффициенты α_k и γ_k определяются рекуррентными формулами

$$\alpha_n = -\frac{d_{n+1}^{(+)}}{\alpha_{n-1} d_{n-1}^{(-)} + d_n},\ \gamma_n = \frac{b_n - \gamma_{n-1} d_{n-1}^{(-)}}{\alpha_{n-1} d_{n-1}^{(-)} + d_n},\ n = 2,3,...,N-1. \qquad (1.24)$$

При этом начальные значения α_1 и γ_1 в соответствии с (1.21) равны

$$\alpha_1 = -\frac{d_2^{(+)}}{d_1},\ \gamma_1 = \frac{b_1}{d_1}. \qquad (1.25)$$

После определения коэффициентов составляющие решения СЛАУ вычисляются по формуле (1.22). При этом значение

$$x_n = \frac{b_N - \gamma_{N-1} d_{N-1}^{(-)}}{\alpha_{N-1} d_{N-1}^{(-)} + d_N} = \gamma_N$$

находится как решение системы из двух уравнений:

$$x_{N-1} = \alpha_{N-1} x_N + \gamma_{N-1},\ d_{N-1}^{(-)} x_{N-1} + d_N x_N = b_N.$$

Отметим, что использование связи между компонентами вектора решения вида (1.11) приводит матрицу СЛАУ к верхней треугольной форме с двумя отличными от нуля диагоналями:

$$\begin{pmatrix} 1 & \alpha_1 & 0 & \dots & \dots & 0 & 0 \\ 0 & 1 & \alpha_2 & \dots & \dots & 0 & 0 \\ 0 & 0 & 1 & \alpha_n & \dots & 0 & 0 \\ \dots & \dots & \dots & 1 & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & \dots & \alpha_{N-2} & 0 \\ 0 & 0 & \dots & \dots & \dots & 1 & \alpha_{N-1} \\ 0 & 0 & \dots & \dots & \dots & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_{N-2} \\ x_{N-1} \\ x_N \end{pmatrix} = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \dots \\ \gamma_{N-2} \\ \gamma_{N-1} \\ \gamma_N \end{pmatrix}.$$

Поэтому описанный алгоритм фактически является модификацией алгоритма исключения. Тем не менее, он широко известен под названием метода прогонки. Функция *PRGN(A,b)* реализует метод в пакете Mathcad.

$$\begin{aligned}
&\text{PRGN(A ,b) :=} \\
&\quad \text{Решение_СЛАУ} \leftarrow \% \\
&\quad \text{методом_прогонки} \leftarrow \% \\
&\quad N \leftarrow \text{rows}(A) - 1 \\
&\quad \alpha_0 \leftarrow -\frac{A_{0,1}}{A_{0,0}} \\
&\quad \gamma_0 \leftarrow \frac{b_0}{A_{0,0}} \\
&\quad \text{for } n \in 1..N-1 \\
&\qquad \alpha_n \leftarrow -\frac{A_{n,n+1}}{\alpha_{n-1} \cdot A_{n,n-1} + A_{n,n}} \\
&\qquad \gamma_n \leftarrow \frac{b_n - \gamma_{n-1} \cdot A_{n,n-1}}{\alpha_{n-1} \cdot A_{n,n-1} + A_{n,n}} \\
&\quad x_N \leftarrow \frac{b_N - \gamma_{N-1} \cdot A_{N,N-1}}{\alpha_{N-1} \cdot A_{N,N-1} + A_{N,N}} \\
&\quad \text{for } n \in N-1..0 \\
&\qquad x_n \leftarrow \alpha_n \cdot x_{n+1} + \gamma_n \\
&\quad x
\end{aligned}$$

При программной реализации метода прогонки выполняется $8N-6$ арифметических операций, что свидетельствует о его высокой вычислительной эффективности.

1.8. Решение СЛАУ методом ортогонализации

В матричной алгебре, помимо уже отмеченных, существует ещё одно чрезвычайно полезное разложение – *QR*-разложение матрицы. Это представление матрицы в виде произведения двух матриц – ортогональной **Q** и верхней треугольной **R**:

$$\mathbf{A}=\mathbf{QR}. \tag{1.26}$$

Существует теорема о том, что любая квадратная матрица может быть представлена в виде разложения (1.26). Если к тому же потребовать, чтобы диагональные элементы матрицы **R** были положительны, то такое разложение будет единственным.

Одним из возможных практических способов построения *QR*-разложения является способ, основанный на преобразовании Хаусхолдера. Он подробно описан в приложении С, где также приведена Mathcad-функция *QR(A)*, реализующая разложение.

Остановимся на использовании *QR*-разложения для решения СЛАУ. С учётом (1.26) система $\mathbf{Ax}=\mathbf{b}$ преобразуется к виду

$$\mathbf{Rx}=\mathbf{Q}^T\mathbf{b}. \tag{1.27}$$

Здесь учтена ортогональность матрицы: $\mathbf{Q}^T\mathbf{Q}=\mathbf{QQ}^T=\mathbf{I}$. СЛАУ (1.27) с верхней треугольной матрицей решается методом обратной подстановки.

Функция *QRSOL(A,b)* служит примером решения СЛАУ методом ортогонализации.

```
QRSOL(A,b) := | Решение_СЛАУ_QR_методом ← %
              | N ← rows(A)
              | QR ← QR(A)
              | Q ← submatrix(QR,0,N − 1,0,N − 1)
              | R ← submatrix(QR,0,N − 1,N,2·N − 1)
              | b ← Q^T·b
              | BS(R,b)
```

Метод ортогонализации, использующий для решения СЛАУ *QR*-разложение матрицы, требует примерно в два раза больше вычислений по сравнению с методом, основанном на *LU*-разложении. Тем не менее, его привлекательность состоит в том, что умножение на ортогональную матрицу

не увеличивает возмущений (погрешностей), которые могут присутствовать в элементах матрицы **A** и вектора **b**. В этом легко убедиться, проведя последовательность преобразований

$$(\mathbf{QA},\mathbf{QA}) = (\mathbf{QA})^T\mathbf{QA} = \mathbf{A}^T\mathbf{Q}^T\mathbf{QA} = \mathbf{A}^T\mathbf{A} = (\mathbf{A},\mathbf{A}),$$

$$(\mathbf{Qb},\mathbf{Qb}) = (\mathbf{Qb})^T\mathbf{Qb} = \mathbf{b}^T\mathbf{Q}^T\mathbf{Qb} = \mathbf{b}^T\mathbf{b} = (\mathbf{b},\mathbf{b}).$$

Широкое применение *QR*-разложение находит в решении матричной проблемы собственных значений.

ГЛАВА 2. ИТЕРАЦИОННЫЕ МЕТОДЫ РЕШЕНИЯ СЛАУ

2.1. Введение в итерационные методы

В отличие от точных, итерационные методы за конечное число арифметических операций позволяют найти лишь приближённое решение системы линейных алгебраических уравнений. Алгоритмы итерационных методов легко конструируются и программируются. Однако для решения вопроса о сходимости того либо иного метода требуется исследование свойств матрицы СЛАУ.

Классические итерационные методы основаны на представлении СЛАУ

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

в эквивалентной форме

$$\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{g}, \tag{2.1}$$

где **B** носит название итерационной матрицы. На основе этой формы итерационный процесс формулируется следующим образом:

1) задаётся вектор начального приближения $\mathbf{x}^{(0)}$;

2) вычисляется последовательность приближений

$$\mathbf{x}^{(k+1)} = \mathbf{B}\mathbf{x}^{(k)} + \mathbf{g}^{(k)},\ k = 0,1,\ldots; \tag{2.2}$$

3) процесс завершается при выполнении неравенства

$$\frac{\left\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right\|}{\left\|\mathbf{x}^{(k+1)}\right\|} \le \varepsilon,$$

определяющего достижение заданной точности ε.

Итерационный процесс (2.2) называется стационарным, если $\mathbf{g}^{(k)} = \mathbf{g} = const$. В противном случае итерационный процесс нестационарен.

Центральным вопросом для процесса (2.2) является вопрос о его сходимости к решению СЛАУ $\mathbf{x}_*$. Для стационарного процесса ответ даёт следующий простой анализ.

Представим *k*-ое приближение в виде

$$\mathbf{x}^{(k)} = \mathbf{x}_* + \mathbf{e}^{(k)}, \tag{2.3}$$

где $\mathbf{e}^{(k)}$ – вектор ошибки *k*-ого приближения. Подставив (2.3) в (2.2), после простых преобразований получим

$$\mathbf{e}^{(k+1)} = \mathbf{B}\mathbf{e}^{(k)} \text{ или } \mathbf{e}^{(k)} = \mathbf{B}^k\mathbf{e}^{(0)}. \tag{2.4}$$

Условием сходимость итерационного процесса (2.2) к $\mathbf{x}_*$ является существование предела

$$\lim_{k\to\infty}\left\|\mathbf{e}^{(k)}\right\| = 0\,. \qquad (2.5)$$

Теперь запишем вектор $\mathbf{e}^{(0)}$ в (2.4) в нормированном базисе собственных векторов $\mathbf{x}_n$ матрицы **B**:

$$\mathbf{e}^{(0)} = \sum_{m=1}^{N} c_m \mathbf{x}_n\,,$$

где c_n – коэффициенты разложения. Тогда вектор ошибки k-ого приближения равен

$$\mathbf{e}^{(k)} = \sum_{n=1}^{N} c_n \mathbf{B}^k \mathbf{x}_n = \sum_{n=1}^{N} c_n \lambda_n^k(\mathbf{B})\mathbf{x}_n\,,$$

где $\lambda_n(\mathbf{B})$ – собственные числа матрицы **B** (см. п. 3.1).

С учётом того, что $\left\|\mathbf{x}_n\right\| = 1$, норма вектора ошибки удовлетворяет неравенству

$$\left\|\mathbf{e}^{(k)}\right\| \le \sum_{n=1}^{N} |\, c_n \,|\, |\lambda_n(\mathbf{B})|^k\,.$$

Следовательно, предел (2.5) имеет место, если

$$s(\mathbf{B}) = \max_n\left(|\lambda_n(\mathbf{B})|\right) < 1\,. \qquad (2.6)$$

Число $s(\mathbf{B})$ называется спектральным радиусом матрицы **B.**

Неравенство (2.6) обычно формулируется в виде основного утверждения о сходимости итерационных методов решения СЛАУ: необходимым и достаточным условием сходимости итерационного процесса является то, что спектральный радиус итерационной матрицы меньше единицы.

Но следует иметь в виду, что вычислительные затраты при нахождении спектрального радиуса матрицы сопоставимы с затратами на решение СЛАУ. Поэтому часто используется более простое достаточное условие сходимости итерационного процесса (2.2), которое следует непосредственно из (2.4):

$$\left\|\mathbf{B}\right\| < 1\,. \qquad (2.7)$$

Можно также показать, что арифметические ошибки в итерационном процессе (2.2) не накапливаются.

2.2. Метод Якоби

Метод Якоби имеет и второе название – метод простой итерации. Его итерационную матрицу можно построить, записав n-ое уравнение СЛАУ в виде

$$a_{nn}x_n = -\sum_{m=1}^{n-1} a_{nm}x_m - \sum_{m=n+1}^{N} a_{nm}x_m + b_n,\ n = 1,2,...,N.$$

В векторно-матричной форме эта запись выглядит как

$$\mathbf{Dx} = -\mathbf{Lx} - \mathbf{Ux} + \mathbf{b}, \tag{2.8}$$

где **L** и **U** – матрицы следующего вида:

$$\mathbf{L} = \begin{pmatrix} 0 & 0 & 0 & ... & 0 \\ a_{21} & 0 & 0 & ... & 0 \\ a_{31} & a_{32} & 0 & ... & 0 \\ ... & ... & ... & ... & ... \\ a_{N1} & a_{N2} & a_{N3} & ... & 0 \end{pmatrix},\ \mathbf{U} = \begin{pmatrix} 0 & a_{12} & a_{13} & ... & a_{1N} \\ 0 & 0 & a_{23} & ... & a_{2N} \\ ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & a_{N-1N} \\ 0 & 0 & 0 & ... & 0 \end{pmatrix},$$

а $\mathbf{D} = diag(a_{11}, a_{22}, ..., a_{NN})$ – диагональная матрица. При этом

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U},$$

Запись (2.8) легко преобразуется к виду

$$\mathbf{x} = -\mathbf{D}^{-1}(\mathbf{A} - \mathbf{D})\mathbf{x} + \mathbf{D}^{-1}\mathbf{b} = (\mathbf{I} - \mathbf{D}^{-1}\mathbf{A})\mathbf{x} + \mathbf{D}^{-1}\mathbf{b}.$$

Следовательно, итерационная матрица **B** и вектор **g** для метода Якоби имеют вид

$$\mathbf{B} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A},\ \mathbf{g} = \mathbf{D}^{-1}. \tag{2.9}$$

С учётом того, что $\mathbf{D}^{-1} = diag(a_{11}^{-1}\, a_{22}^{-1}, ..., a_{NN}^{-1})$, **B** и **g** легко вычисляются.

Пример вычислений матрицы **B** с использованием аналитического режима Mathcad позволяет выяснить условие сходимости метода Якоби.

$$i := 0..2 \qquad d_i := a_{i+1,i+1} \qquad F(i,j) := a_{i,j}$$

$$I := identity(3) \qquad D := diag(d) \qquad A := matrix(3,3,F)$$

$$A \rightarrow \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \qquad D \rightarrow \begin{pmatrix} a_{1,1} & 0 & 0 \\ 0 & a_{2,2} & 0 \\ 0 & 0 & a_{3,3} \end{pmatrix}$$

$$\left(I - D^{-1}\cdot A\right) \rightarrow \begin{pmatrix} 0 & \frac{-1}{a_{1,1}}\cdot a_{1,2} & \frac{-1}{a_{1,1}}\cdot a_{1,3} \\ \frac{-1}{a_{2,2}}\cdot a_{2,1} & 0 & \frac{-1}{a_{2,2}}\cdot a_{2,3} \\ \frac{-1}{a_{3,3}}\cdot a_{3,1} & \frac{-1}{a_{3,3}}\cdot a_{3,2} & 0 \end{pmatrix}$$

В приведенном примере неравенство (2.7) для равномерной нормы матрицы $\|\mathbf{B}\|_i$ (см. приложение А) выполняется, если

$$\sum_{m\neq n}^{3} |a_{nm}| < |a_{nn}|,\ n = 1,2,3. \qquad (2.10)$$

Матрица со свойством (2.10) называется матрицей со строгим диагональным преобладанием. Это утверждение о достаточности строгого диагонального преобладания матрицы СЛАУ для сходимости метода Якоби, естественно, обобщается на матрицы произвольного размера $N \times N$:

$$\sum_{m\neq n}^{N} |a_{nm}| < |a_{nn}|,\ n = 1,2,...,N.$$

```
YacIter(A, b, ε) := | Решение_СЛАУ_методом_Якоби ← %
                    | N ← rows(A)
                    | for n ∈ 0.. N − 1
                    |   d_n ← (A_n,n)^−1
                    | Din ← diag(d)
                    | B ← identity(N) − Din·A
                    | g ← Din·b
                    | k ← 1
                    | x^⟨0⟩ ← b
                    | while k < 300
                    |     | x^⟨k⟩ ← B·x^⟨k−1⟩ + g
                    |     | return x^⟨k⟩ if |x^⟨k⟩ − x^⟨k−1⟩| < ε
                    |     | k ← k + 1
                    | k
```

Итерации по алгоритму (2.2) с матрицей и вектором (2.9) реализуются Mathcad-функцией *YacIter(A,b,ε)*.

2.3. Метод Зейделя

В методе Зейделя, представляющем собой простую модификацию метода Якоби, алгоритм итераций записывается в виде

$$a_{11}x_1^{(k+1)} = -\sum_{m=2}^{N} a_{1m}x_m^{(k)} + b_1;$$

$$a_{nn}x_n^{(k+1)} = -\sum_{m=1}^{n-1} a_{nm}x_m^{(k+1)} - \sum_{m=n+1}^{N} a_{nm}x_m^{(k)} + b_n,\ n = 2,3,...,N-1;. \qquad (2.11)$$

$$a_{NN}x_N^{(k+1)} = -\sum_{m=1}^{N-1} a_{Nm}x_m^{(k+1)} + b_N.$$

Здесь на каждой итерации для вычисления *k+1*-го приближения для компоненты вектора решения x_n предлагается использовать уже найденные *k+1*-ые приближения для компонент $x_1, x_2, ..., x_{n-1}$. Оставшиеся компоненты суммируются, как и в методе Якоби.

В векторно-матричной форме итерации (2.11) выглядят как

$$\mathbf{D}\mathbf{x}^{(k+1)} = -\mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{U}\mathbf{x}^{(k)} + \mathbf{b},$$

или как

$$(\mathbf{D}+\mathbf{L})\mathbf{x}^{(k+1)} = -\mathbf{U}\mathbf{x}^{(k)} + \mathbf{b}. \qquad (2.12)$$

Отсюда следуют выражения для итерационной матрицы **B** и вектора **g** метода Зейделя:

$$\mathbf{B} = -(\mathbf{D}+\mathbf{L})^{-1}\mathbf{U},\ \mathbf{g}^{(k)} = \mathbf{g} = (\mathbf{D}+\mathbf{L})^{-1}\mathbf{b}. \qquad (2.13)$$

При практических вычислениях обращать матрицу $\mathbf{D}+\mathbf{L}$ не имеет смысла, т.к. большую вычислительную эффективность демонстрирует алгоритм в форме (2.12). $\mathbf{D}+\mathbf{L}$ – нижняя треугольная матрица, поэтому СЛАУ (2.12) легко решается обратной подстановкой.

Выполнение условия (2.7) для нормы матрицы **B** (2.13), как правило, проверить достаточно сложно. Поэтому на практике считается, что сходимость метода Зейделя обеспечивается, если сходится метод Якоби.

Следует отметить, что итерации (2.12) предполагают, что вычисление компонент решения СЛАУ проводится в последовательности $x_1 \to x_N$. При обратной последовательности ($x_N \to x_1$) итерации проводятся по алгоритму

$$(\mathbf{D}+\mathbf{U})\mathbf{x}^{(k+1)} = -\mathbf{L}\mathbf{x}^{(k)} + \mathbf{b}. \qquad (2.14)$$

Теоретически итерации (2.12) и (2.14) равнозначны и требуют одинаковых вычислительных затрат. Функция *SeiIter(A,b,ε)* реализует в Mathcad вычисления по алгоритму (2.12).

```
SeiIter(A,b,ε) := | Решение_СЛАУ_методом_Зейделя ← %
                  | N ← rows(A) – 1
                  | U ← A
                  | for n ∈ 0..N
                  |   for m ∈ 0..n
                  |     U[n,m] ← 0
                  | DL ← A – U
                  | k ← 1
                  | y^⟨0⟩ ← b
                  | while k < 300
                  |   y^⟨k⟩ ← ls(DL, b – U·y^⟨k–1⟩)
                  |   return stack(y^⟨k⟩, k) if |y^⟨k⟩ – y^⟨k–1⟩| < ε
                  |   k ← k + 1
                  | k
```

На рис. 2.1 показаны графики зависимостей норм векторов решения СЛАУ с матрицей **A** (она обозначена на рисунке) и вектором $\mathbf{b}^T = (0,0,1,0)$ от числа итераций, проведённых по методу Якоби (МЯ) и методу Зейделя (МЗ).

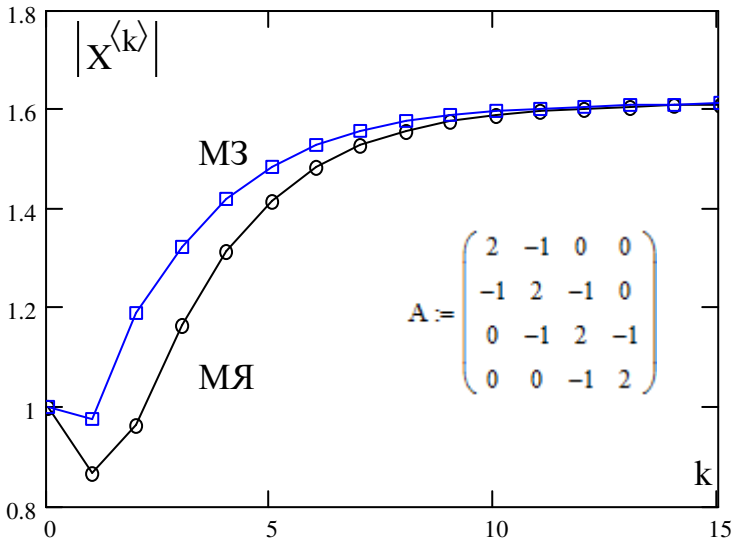

Рис. 2.1. Зависимость нормы вектора решения от числа итераций

Как следует из рисунка итерационные процессы для матрицы **A** – матрицы со слабым диагональным преобладанием – сходятся к единственному точному решению $X^T = (0.4, 0.8, 1.2, 0.6)$:

$$YacIter(A, b, 0.001)^T = (\ 0.3995 \quad 0.7997 \quad 1.1992 \quad 0.5998\ )$$

Дополнительно отметим, что вычисления спектрального радиуса итерационной матрицы **B** в методах Якоби и Зейделя дают следующие результаты.

$$BY := I - D^{-1} \cdot A \qquad \lambda(BY) := eigenvals(BY)$$

$$max\left(\left|min\left(\lambda(BY)\right)\right|, \left|max\left(\lambda(BY)\right)\right|\right) = 0.809$$

$$BS := -(A - U)^{-1} \cdot U \quad \lambda(BS) := eigenvals(BS)$$

$$max\left(\left|min\left(\lambda(BS)\right)\right|, \left|max\left(\lambda(BS)\right)\right|\right) = 0.655$$

Меньший спектральный радиус итерационной матрицы метода Зейделя обеспечивает его более быструю сходимость по сравнению с методом Якоби.

2.4. Методы релаксации

Графики, показанные на рис. 2.1, допускают простую физическую интерпретацию. Если связать с выполнением каждой из итераций определённый отрезок времени, то можно считать, что за исключением начальных участков графики отображают процессы релаксации динамических систем к состоянию равновесия. Время релаксации в методах Якоби и Зейделя пропорционально спектральному радиусу матрицы **B**.

Для того, чтобы сформировать итерационный процесс с управляемым временем релаксации τ, проанализируем динамику простейшей физической системы с математической моделью в виде обыкновенного дифференциального уравнения первого порядка:

$$\frac{dU}{dt} = -\frac{1}{\tau}U + \frac{1}{\tau}E. \tag{2.15}$$

Примером такой системы может служить *RC*-цепь с конденсатором *C*, заряжаемым от источника эдс *E* через сопротивление *R*. В этом случае *U* – это напряжение на конденсаторе, а время релаксации (постоянная времени) вычисляется как $\tau = RC$.

Численное интегрирование задачи Коши для уравнения (2.15) на дискретной временной сетке $t_k = k\Delta t$ методом Эйлера даёт следующую рекуррентную формулу:

$$U^{(k+1)} = \left(1 - \frac{\Delta t}{\tau}\right)U^{(k)} + \frac{\Delta t}{\tau}E,\ k = 0,1,... \tag{2.16}$$

Значение $U^{(0)}$ в этой формуле – начальное условие задачи Коши. Без ограничения общности шаг интегрирования в (2.16) можно считать единичным: $\Delta t = 1$. С учётом этого и, введя в рассмотрение частоту релаксации $\omega = 1/\tau$, запишем формулу (2.16) в виде

$$U^{(k+1)} = (1-\omega)U^{(k)} + \omega E,\ k = 0,1,... \qquad (2.17)$$

При численном исследовании процесса зарядки конденсатора на основе формулы (2.17) формируется итерационный процесс $U^{(0)} \to U^{(1)} \to U^{(2)} \to ...$, сходящийся к E при условии, что $0 < \omega < 2$.

В многомерной системе процесс релаксации (2.17) принимает вид

$$\mathbf{x}^{(k+1)} = (1-\omega)\mathbf{x}^{(k)} + \omega\mathbf{e}. \qquad (2.18)$$

Сопоставим его с итерационным процессом метода Якоби

$$\mathbf{x}^{(k+1)} = \mathbf{B}\mathbf{x}^{(k)} + \mathbf{g}$$

и проведём объединение процессов. Получим

$$\mathbf{x}^{(k+1)} = (1-\omega)\mathbf{x}^{(k)} + \omega\mathbf{B}\mathbf{x}^{(k)} + \omega\mathbf{g},\ k = 0,1,... \qquad (2.19)$$

Отметим, что при частоте релаксации $\omega = 1$ алгоритм (2.19) переходит в алгоритм метода Якоби.

С учётом того, что итерационная матрица **B** и вектор **g** определяются выражениями (2.9), в развёрнутом виде алгоритм релаксации (2.19) выглядит как

$$\mathbf{x}^{(k+1)} = \left(\mathbf{I} - \omega\mathbf{D}^{-1}\mathbf{A}\right)\mathbf{x}^{(k)} + \omega\mathbf{D}^{-1}\mathbf{b},\ k = 0,1,... \qquad (2.20)$$

Теперь, варьируя частоту релаксации ω, можно в определённых пределах управлять скоростью сходимости итерационного процесса. При $0 < \omega < 1$ релаксация носит название нижней, а при $1 < \omega < 2$ – верхней.

На рис. 2.2 показан график зависимости спектрального радиуса итерационной матрицы в алгоритме (2.20) для СЛАУ с матрицей **A** (обозначена на рисунке) от частоты релаксации. Минимум значения спектрального радиуса $s_{\min} = 0.334$ достигается при значении $\omega = 0.888$. В методе Якоби $s = 0.5$ (точка на графике).

На рис. 2.3 показаны отклонения норм векторов решения СЛАУ с матрицей **A** (она представлена на рисунке) и вектором $\mathbf{b}^T = (0,0,1)$ от нормы точного решения в зависимости от числа итераций, проведённых методом Якоби (МЯ) и методом релаксации (МР). Меньший спектральный радиус итерационной матрицы в расчётах методом релаксации (в данном примере – нижней) обеспечивает ему более быструю сходимость, чем методу Якоби. Для достижения относительной точности в три значащих цифры методу Якоби требуются 17 итераций, методу релаксации – 11 итераций. Таким

образом, заключение об ускорении сходимости путём подбора частоты релаксации подтверждается.

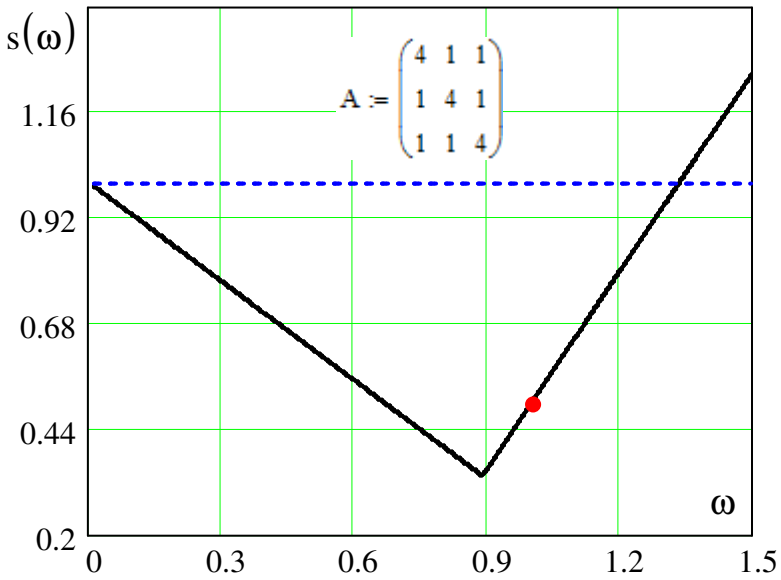

Рис. 2.2. Зависимость спектрального радиуса итерационной матрицы от частоты релаксации

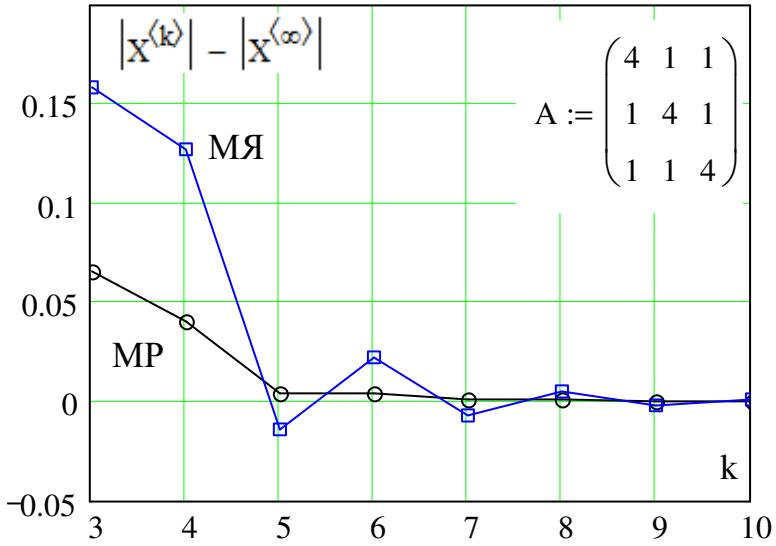

Рис. 2.3. Зависимости норм векторов решений от числа итераций

Перейдём к разработке метода релаксации, основанного на методе Зейделя. Запишем итерационный процесс (2.11) в виде

$$\mathbf{x}^{(k+1)} = -\mathbf{D}^{-1}\mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{D}^{-1}\mathbf{U}\mathbf{x}^{(k)} + \mathbf{D}^{-1}\mathbf{b}$$

и объединим его с процессом релаксации (2.18). В результате объединения получим процесс последовательной релаксации вида

$$\mathbf{x}^{(k+1)} = (1-\omega)\mathbf{x}^{(k)} - \omega\mathbf{D}^{-1}\mathbf{L}\mathbf{x}^{(k+1)} - \omega\mathbf{D}^{-1}\mathbf{U}\mathbf{x}^{(k)} + \omega\mathbf{D}^{-1}\mathbf{b}\,.$$

После несложных преобразований практически реализуемый алгоритм метода последовательной релаксации (successive relaxation) записывается как

$$(\mathbf{D}+\omega\mathbf{L})\mathbf{x}^{(k+1)} = [\mathbf{D}-\omega(\mathbf{A}-\mathbf{L})]\mathbf{x}^{(k)} + \omega\mathbf{b}\,. \qquad (2.21)$$

Функция *SOR(A,b,ω,ε)* реализует в Mathcad вычисления по алгоритму (2.21).

```
SOR(A, b, ω, ε) := | Решение_СЛАУ_методом ← %
                   | последовательной_релаксации ← %
                   | N ← rows(A) − 1
                   | L ← A
                   | for n ∈ 0.. N
                   |   for m ∈ n.. N
                   |     L_n,m ← 0
                   |     D_n,n ← A_n,n
                   | DL ← D + ω·L
                   | DAL ← D − ω·(A − L)
                   | k ← 1
                   | y^⟨0⟩ ← b
                   | while k < 300
                   |   y^⟨k⟩ ← ls(DL, ω·b + DAL·y^⟨k−1⟩)
                   |   return y if |y^⟨k⟩ − y^⟨k−1⟩| < ε
                   |   k ← k + 1
                   | k
```

Выражение для итерационной матрицы метода последовательной релаксации легко получить из (2.21):

$$\mathbf{B}_{sr} = (\mathbf{D}+\omega\mathbf{L})^{-1}(\mathbf{D}-\omega(\mathbf{A}-\mathbf{L}))\ . \qquad (2.22)$$

Варьируя частоту релаксации ω, можно управлять скоростью сходимости итерационного процесса (2.21). На рис. 2.4 показан график зависимости спектрального радиуса итерационной матрицы (2.22) от частоты релаксации (для СЛАУ с матрицей **A**, обозначенной на рисунке).

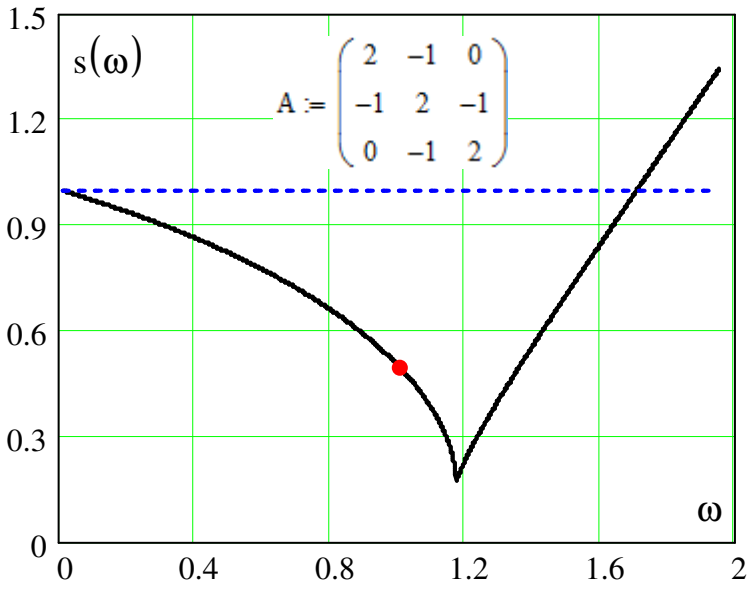

Рис. 2.4. Зависимость спектрального радиуса матрицы $\mathbf{B}_{sr}$ *от частоты релаксации*

Минимум значения спектрального радиуса $s_{\min} = 0.172$ достигается при значении $\omega_{opt} = 1.172$. В методе Зейделя $s = 0.5$ (точка на графике). Это указывает на более быструю сходимость метода последовательной релаксации (в данном примере – верхней), чем метода Зейделя.

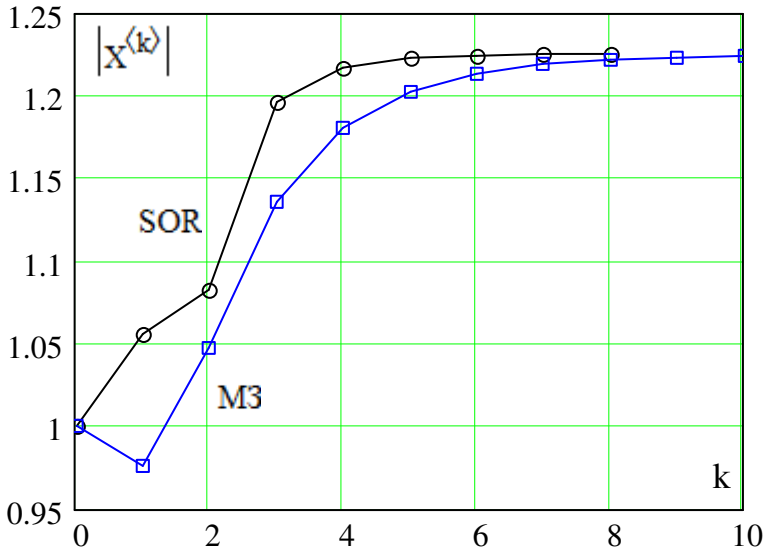

Рис. 2.5. Зависимости норм векторов решений от числа итераций

На рис. 2.5 показаны нормы векторов решения СЛАУ с матрицей **A** и вектором $\mathbf{b}^T = (0,1,0)$ в зависимости от числа итераций, проведённых методом Зейделя (МЗ) и методом последовательной верхней релаксации

(SOR – successive over relaxation) с ω_{opt}. Для достижения относительной точности в три значащих цифры методу Зейделя требуются 13 итераций, методу последовательной верхней релаксации – 8 итераций.

2.5. Метод минимальных невязок

Описанные выше итерационные методы не обладают гарантированной сходимостью, а анализ сходимости базируется на оценках спектрального радиуса итерационной матрицы. Причём, оценки спектрального радиуса зачастую требуют вычислительных затрат, сравнимых с затратами на собственно решение СЛАУ. Поэтому практический интерес представляют методы, сходимость которых обеспечивается вариациями параметров итерационных процессов. Одним из них является метод минимальных невязок.

По определению вектор невязки СЛАУ – это вектор

$$\mathbf{R}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}. \tag{2.23}$$

Очевидно, что для вектора решения СЛАУ $\mathbf{x}_r$ вектор невязки принимает нулевое значение: $\mathbf{R}(\mathbf{x}_r) = 0$.

Считая, что вектор *k*-го приближения и вектор решения различаются малым приращением ξ, так что

$$\mathbf{x}_r = \mathbf{x}^{(k)} + \xi,$$

разложим вектор невязки (2.23) в ряд Тейлора в окрестности $\mathbf{x}_r$:

$$\mathbf{R}(\mathbf{x}_r) = \mathbf{R}(\mathbf{x}^{(k)}) + \mathbf{J}(\mathbf{x}^{(k)})\xi = 0. \tag{2.24}$$

Здесь $\mathbf{J}(\mathbf{x}^{(k)})$ – матрица Якоби с элементами $J_{nm} = \partial R_n / \partial x_m$, вычисляемыми для вектора $\mathbf{x}^{(k)}$. Слагаемые второй и всех следующих степеней по компонентам приращения ξ в разложении (2.24) отсутствуют ввиду линейности невязки (2.23), как функции вектора $\mathbf{x}$.

Уравнение (2.24) имеет формальное решение относительно вектора приращений

$$\xi = -\mathbf{J}^{-1}(\mathbf{x}^{(k)})\mathbf{R}(\mathbf{x}^{(k)}). \tag{2.25}$$

Но фактически это решение не имеет никаких преимуществ перед решением $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ исходной СЛАУ, тем более что для вектора невязки (2.23) $\mathbf{J}(\mathbf{x}^{(k)}) = \mathbf{A}$.

Тогда можно предложить другой подход – замену матричного умножения в формуле (2.25) скалярным умножением на параметр τ_k, связанный с матрицей $\mathbf{A}$:

$$\xi = -\tau_k \mathbf{R}(\mathbf{x}^{(k)}). \tag{2.26}$$

При использовании такой замены сумма $\mathbf{x}^{(k)} + \xi$ уже не дает решения СЛАУ, но её можно использовать для вычисления *k+1*-го приближения к решению:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \tau_k \mathbf{R}(\mathbf{x}^{(k)}). \tag{2.27}$$

Теперь, если определить способ вычисления параметра τ_k, рекуррентная формула (2.27) будет определять нестационарный итерационный процесс приближений к решению $\mathbf{x}_r$ СЛАУ.

Так как переход от (2.25) к (2.26) пока не был сопряжён с какими либо требованиями к итерационному параметру τ_k, то теперь их можно сформулировать как условие минимальности модуля вектора невязки $\mathbf{R}(\mathbf{x}^{(k+1)})$. В дальнейшем для упрощения записи выражений будем использовать обозначения вида $\mathbf{R}_k = \mathbf{R}(\mathbf{x}^{(k)})$. Выразим $|\mathbf{R}_{k+1}|^2$ через вектор невязки *k*-го приближения. Предварительно, используя формулу (2.27), найдём связь векторов $\mathbf{R}_k$ и $\mathbf{R}_{k+1}$:

$$\mathbf{R}_{k+1} = \mathbf{A}\mathbf{x}^{(k+1)} - \mathbf{b} = \mathbf{A}\left(\mathbf{x}^{(k)} - \tau_k \mathbf{R}_k\right) - \mathbf{b} = \left(\mathbf{A}\mathbf{x}^{(k)} - \mathbf{b}\right) - \tau_k \mathbf{A}\mathbf{R}_k = \mathbf{R}_k - \tau_k \mathbf{A}\mathbf{R}_k$$

Теперь вычислим $|\mathbf{R}_{k+1}|^2 = (\mathbf{R}_k - \tau_k \mathbf{A}\mathbf{R}_k, \mathbf{R}_k - \tau_k \mathbf{A}\mathbf{R}_k)$:

$$|\mathbf{R}_{k+1}|^2 = |\mathbf{R}_k|^2 - 2\tau_k (\mathbf{R}_k, \mathbf{A}\mathbf{R}_k) + \tau_k^2 |\mathbf{A}\mathbf{R}_k|^2 = f(\tau_k). \tag{2.28}$$

Минимальность $|\mathbf{R}_{k+1}|$ достигается при значении τ_k, удовлетворяющем уравнению $f'(\tau_k) = 0$, при условии, что $f''(\tau_k) > 0$. Последнее неравенство всегда выполняется, т.к.

$$f''(\tau_k) = 2|\mathbf{A}\mathbf{R}_k|^2 > 0.$$

Поэтому $|\mathbf{R}_{k+1}| = \min$, если

$$\tau_k = \frac{(\mathbf{R}_k, \mathbf{A}\mathbf{R}_k)}{|\mathbf{A}\mathbf{R}_k|^2}. \tag{2.29}$$

Итерационный алгоритм (2.27) и (2.29) составляет основное содержание метода минимальных невязок. Метод реализуется Mathcad-функцией *MinDiscrep(A,b,ε)*.

Следует ещё убедиться в сходимости итераций (2.27) при условии, что итерационный параметр определяется выражением (2.29). Для этого подставим τ_k из (2.29) в выражение (2.28). Получим

$$|\mathbf{R}_{k+1}|^2 = |\mathbf{R}_k|^2 - \frac{(\mathbf{R}_k, \mathbf{A}\mathbf{R}_k)^2}{|\mathbf{A}\mathbf{R}_k|^2}.$$

Вследствие положительности каждого из слагаемых в правой части этого выражения $|\mathbf{R}_{k+1}|^2 < |\mathbf{R}_k|^2$, что доказывает сходимость итерационного процесса. Правда, скорость сходимости часто невелика, что демонстрирует рис. 2.6, на котором показаны зависимости от номера итерации модулей векторов приближений к решению СЛАУ методом минимальных невязок (ММН) и методом Зейделя (МЗ). Матрица системы и вектор её правых частей представлены на рисунке.

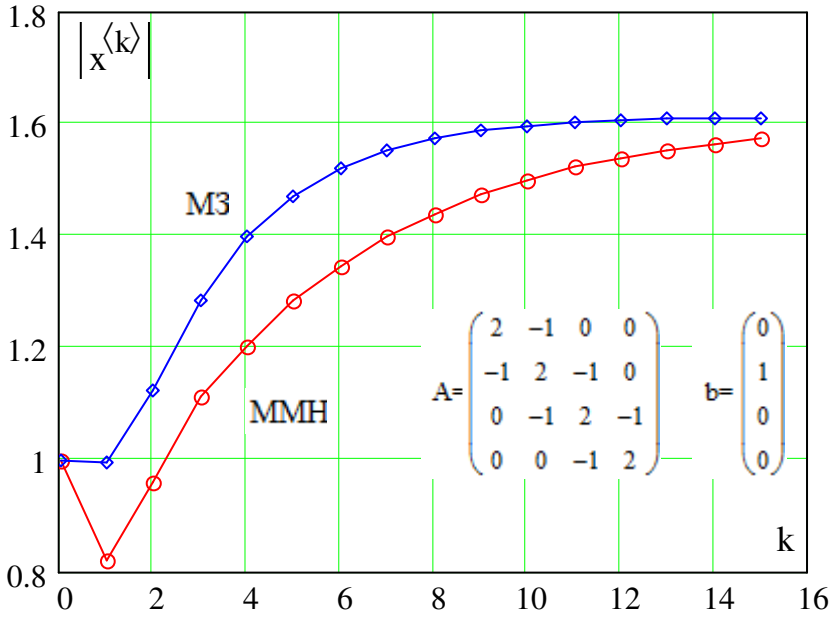

Рис. 2.6. Модули векторов приближений к решению СЛАУ

На рис. 2.7 представлено сечение плоскостью $x_1 x_6$ восьмимерного пространства векторов системы $\mathbf{Ax} = \mathbf{b}$ с трёхдиагональной матрицей, содержащей отличные от нуля элементы $d = 2$ на главной диагонали и $d_{\pm} = -1$ на верхней и нижней поддиагоналях. Вектор правых частей имеет отличные от нуля компоненты $b_5 = -1$ и $b_6 = b_7 = 1$. Точки соответствуют значениям $\left(x_1^{(k)}, x_6^{(k)}\right)$, полученным в результате k-ой итерации методом минимальных невязок или методом Зейделя.

Большие расстояния между точками последовательных итераций свидетельствуют о более высокой скорости сходимости. Значения компонент решения $x_1 = 0.111$, $x_6 = 0.167$ достигаются методом Зейделя за 65 итераций, в то время как методом минимальных невязок – за 116 итераций.

Таким образом, метод минимальных невязок, обеспечивая гарантированную сходимость итерационного процесса, демонстрирует, как правило, более медленную скорость сходимости, чем метод Зейделя или методы релаксации.

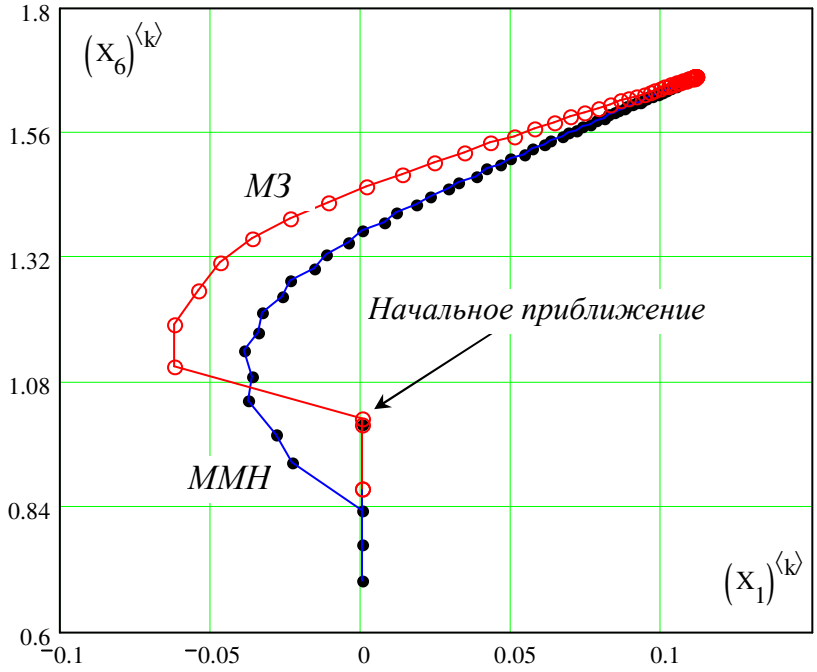

Рис. 2.7. Сечение многомерного пространства плоскостью $x_1 x_6$

MinDiscrep(A, b, ε) :=

Метод_минимальных_невязок_для_СЛАУ ← %

$x^{\langle 0\rangle} \leftarrow b$

$R^{\langle 0\rangle} \leftarrow A \cdot x^{\langle 0\rangle} - b$

$$\tau_0 \leftarrow \frac{\left(\left(R^{\langle 0\rangle}\right)^T \cdot A \cdot R^{\langle 0\rangle}\right)_0}{\left(\left|A \cdot R^{\langle 0\rangle}\right|\right)^2}$$

$k \leftarrow 1$

while 1

$x^{\langle k\rangle} \leftarrow x^{\langle k-1\rangle} - \tau_{k-1} \cdot R^{\langle k-1\rangle}$

$R^{\langle k\rangle} \leftarrow A \cdot x^{\langle k\rangle} - b$

$$\tau_k \leftarrow \frac{\left(\left(R^{\langle k\rangle}\right)^T \cdot A \cdot R^{\langle k\rangle}\right)_0}{\left(\left|A \cdot R^{\langle k\rangle}\right|\right)^2}$$

return $x^{\langle k\rangle}$ if $\left|x^{\langle k\rangle} - x^{\langle k-1\rangle}\right| < \varepsilon$

$k \leftarrow k + 1$

k

2.6. Метод наискорейшего спуска

Метод наискорейшего спуска широко известен как метод минимизации функций многих переменных $F(\mathbf{x})$ – целевой функции. Алгоритм этого метода определяется в виде последовательности смещений

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - h_k \nabla F(\mathbf{x}^{(k)}) \tag{2.30}$$

в направлении, противоположном направлению градиента $\nabla F(\mathbf{x})$. При этом шаг смещения h_k на каждой итерации определяется из условия минимума функции

$$\varphi(h_k) = F\left(\mathbf{x}^{(k)} - h_k \nabla F(\mathbf{x}^{(k)})\right).$$

Для применения этого метода к решению СЛАУ необходимо соответствующим образом определить целевую функцию. Сделаем это с помощью вектора ошибки

$$\mathbf{e}(\mathbf{x}) = \mathbf{x} - \mathbf{x}_r, \tag{2.31}$$

где $\mathbf{x}_r$ – вектор решения СЛАУ.

В дальнейшем будем считать матрицу $\mathbf{A}$ симметричной и положительно определённой (*СПО*-матрицей) и рассмотрим функцию

$$F(\mathbf{x}) = \left(\mathbf{e}(\mathbf{x}), \mathbf{A}\mathbf{e}(\mathbf{x})\right). \tag{2.32}$$

Вследствие положительной определённости матрицы $F(\mathbf{x}) > 0$ для любого вектора $\mathbf{e} \neq 0$ и принимает нулевое значение при $\mathbf{e} = 0$, т.е. при $\mathbf{x} = \mathbf{x}_r$. Таким образом, решение задачи о нулевом минимуме функции (2.32) даёт решение СЛАУ $\mathbf{A}\mathbf{x} = \mathbf{b}$.

Проведём преобразования, необходимые для записи итерационного алгоритма (2.30) в явной форме. С учётом определения (2.31) получим

$$F(\mathbf{x}) = \left(\mathbf{x}, \mathbf{A}\mathbf{x}\right) + \left(\mathbf{x}_r, \mathbf{A}\mathbf{x}_r\right) - 2\left(\mathbf{x}, \mathbf{A}\mathbf{x}_r\right) = \left(\mathbf{x}, \mathbf{A}\mathbf{x}\right) - 2\left(\mathbf{x}, \mathbf{b}\right) + \left(\mathbf{x}_r, \mathbf{b}\right). \tag{2.33}$$

При этом использовано свойство симметрии матрицы $\mathbf{A}$. Форма (2.33) позволяет легко вычислить градиент

$$\nabla F(\mathbf{x}) = 2\left(\mathbf{A}\mathbf{x} - \mathbf{b}\right) = 2\mathbf{R}(\mathbf{x}) \tag{2.34}$$

и записать выражение

$$\begin{aligned} \varphi(h_k) = &\left(\mathbf{x}^{(k)} - 2h_k \mathbf{R}(\mathbf{x}^{(k)}), \mathbf{A}(\mathbf{x}^{(k)} - 2h_k \mathbf{R}(\mathbf{x}^{(k)}))\right) - \\ &- 2\left(\mathbf{x}^{(k)} - 2h_k \mathbf{R}(\mathbf{x}^{(k)}), \mathbf{b}\right) + \left(\mathbf{x}_r, \mathbf{b}\right). \end{aligned} \tag{2.35}$$

Отметим, что функция $\varphi(h_k)$ представляет собой квадратичную параболу, поэтому одномерная минимизация выполняется точно. Условие минимума $\varphi'(h_k) = 0$ после дифференцирования выражения (2.35) приводит к уравнению

$$2h_k\left(\mathbf{R}(\mathbf{x}^{(k)}),\mathbf{AR}(\mathbf{x}^{(k)})\right)-\left(\mathbf{R}(\mathbf{x}^{(k)}),\mathbf{R}(\mathbf{x}^{(k)})\right)=0,$$

из которого следует, что шаг смещения в нулевой минимум функции (2.33) на *k*-ой итерации равен

$$h_k=\frac{1}{2}\frac{\left|\mathbf{R}(\mathbf{x}^{(k)})\right|^2}{(\mathbf{R}(\mathbf{x}^{(k)}),\mathbf{AR}(\mathbf{x}^{(k)}))}. \tag{2.36}$$

Подставляя выражение градиента (2.34) и шаг смещения (2.36) в общую формулу (2.30), получим итерационный алгоритм наискорейшего спуска для решения СЛАУ

$$\mathbf{x}^{(k+1)}=\mathbf{x}^{(k)}-\frac{\mathbf{R}(\mathbf{x}^{(k)})\left|\mathbf{R}(\mathbf{x}^{(k)})\right|^2}{(\mathbf{R}(\mathbf{x}^{(k)}),\mathbf{AR}(\mathbf{x}^{(k)}))}. \tag{2.37}$$

Алгоритм (2.37) реализуется Mathcad-функцией *MSD(A,b,ε)*.

$\mathrm{MSD}(A,b,\varepsilon) :=$ | Метод_наискорейшего_спуска_для_СЛАУ ← %
| $x^{\langle 0\rangle} \leftarrow b$
| $R^{\langle 0\rangle} \leftarrow A\cdot x^{\langle 0\rangle} - b$
| $h_0 \leftarrow \dfrac{\left(\left|R^{\langle 0\rangle}\right|\right)^2}{\left(\left(R^{\langle 0\rangle}\right)^{T}\cdot A\cdot R^{\langle 0\rangle}\right)_0}$
| $k \leftarrow 1$
| while 1
| | $x^{\langle k\rangle} \leftarrow x^{\langle k-1\rangle} - h_{k-1}\cdot R^{\langle k-1\rangle}$
| | $R^{\langle k\rangle} \leftarrow A\cdot x^{\langle k\rangle} - b$
| | $h_k \leftarrow \dfrac{\left(\left|R^{\langle k\rangle}\right|\right)^2}{\left(\left(R^{\langle k\rangle}\right)^{T}\cdot A\cdot R^{\langle k\rangle}\right)_0}$
| | return x if $\left|x^{\langle k\rangle} - x^{\langle k-1\rangle}\right| < \varepsilon$
| | $k \leftarrow k + 1$
| k

Фактически функция *MSD* отличается от функции *MinDiscrep*, реализующей метод минимальных невязок, лишь способом вычисления параметров итерационных процессов τ_k.

На рис. 2.8 показано сечение плоскостью x_1x_6 восьмимерного пространства векторов системы $\mathbf{Ax}=\mathbf{b}$ с трёхдиагональной матрицей, содержащей отличные от нуля элементы $d=2$ на главной диагонали и $d_{\pm}=-1$ на верхней и нижней поддиагоналях. Вектор правых частей имеет отличные от нуля компоненты $b_5=-1$ и $b_6=b_7=1$. Точки соответствуют значениям $\left(x_1^{(k)}, x_6^{(k)}\right)$, полученным в результате *k*-ой итерации наискорейшего спуска или методом минимальных невязок.

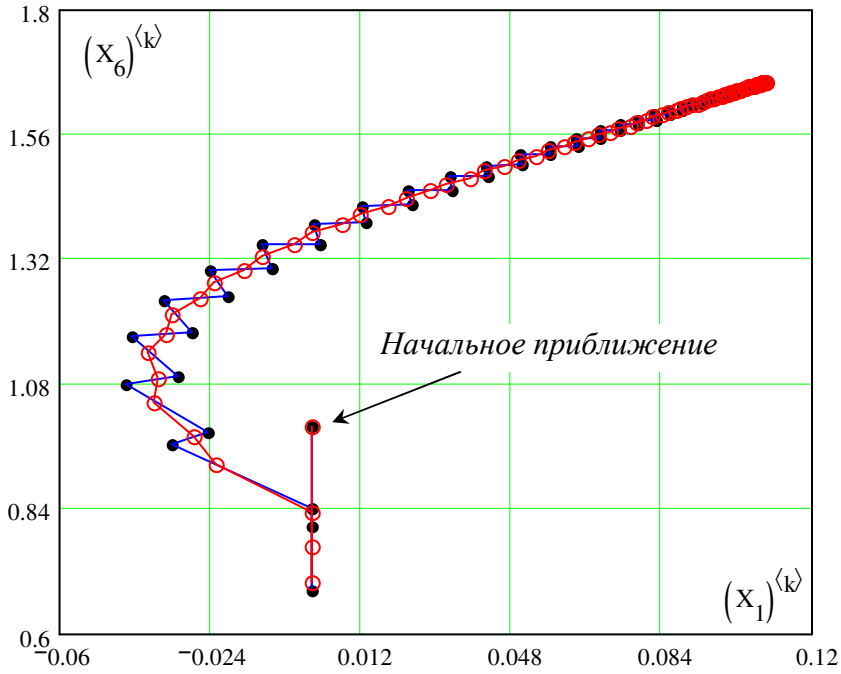

Рис. 2.8. Сечение многомерного пространства плоскостью x_1x_6

Траектория наискорейшего спуска характеризуется затухающими «осцилляциями», в отличие от достаточно плавной траектории метода минимальных невязок. В данном примере вычислительная эффективность двух сопоставляемых методов примерно одинакова: значения компонент решения $x_1=0.111$, $x_6=0.167$ достигаются методом наискорейшего спуска за 120 итераций, а методом минимальных невязок – за 116 итераций. Этот вывод о примерно равной эффективности методов минимальных невязок и наискорейшего спуска в целом справедлив для *СПО*-матриц.

ГЛАВА 3. МАТРИЧНАЯ ПРОБЛЕМА СОБСТВЕННЫХ ЗНАЧЕНИЙ

3.1. Физическое введение в матричную проблему собственных значений

Рассмотрим задачу о колебаниях в системе двух связанных пружинных маятников. Схематически эта колебательная система изображена на рис. 3.1.

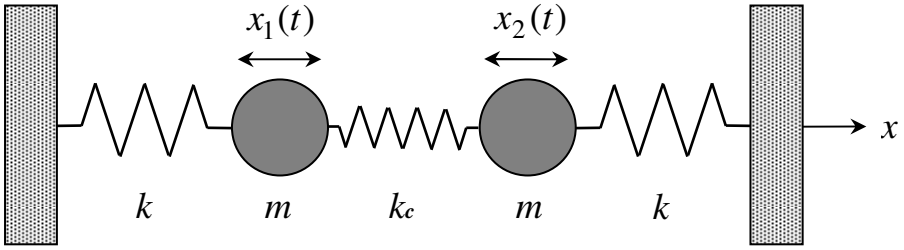

Рис. 3.1.Связанные пружинные маятники

В состоянии равновесия смещения x_1 и x_2 грузов массами m равны нулю. В динамическом состоянии смещения подчиняются системе дифференциальных уравнений, математически отражающих второй закон Ньютона:

$$\begin{aligned} m\frac{d^2x_1}{dt^2} &= -kx_1 + k_c(x_2 - x_1), \\ m\frac{d^2x_2}{dt^2} &= -kx_2 + k_c(x_1 - x_2). \end{aligned} \tag{3.1}$$

Здесь k и k_c – жесткости основных пружин и пружины связи. Введя стандартные обозначения ω_0 для собственной частоты невозмущённого осциллятора κ для параметра связи

$$\omega_0 = \sqrt{\frac{k}{m}} \text{ и } \kappa = \frac{k_c}{k},$$

систему уравнений (3.1) запишем в виде

$$\begin{aligned} \frac{d^2x_1}{dt^2} &= -\omega_0^2(1+\kappa)x_1 + \kappa\omega_0^2 x_2, \\ \frac{d^2x_2}{dt^2} &= -\omega_0^2(1+\kappa)x_2 + \kappa\omega_0^2 x_1. \end{aligned} \tag{3.2}$$

Решение системы (3.2) ищем в форме гармонических колебаний

$$x_1(t) = X_1\exp(j\omega t),\ x_2(t) = X_2\exp(j\omega t).$$

На физическом уровне такой вид решения обоснован тем, что в системе связанных осцилляторов колебания происходят с единой частотой ω, но, возможно, с разными комплексными амплитудами X_1 и X_2. После дифференцирования и сокращения на экспоненту, система дифференциальных уравнений (3.2) преобразуется в систему алгебраических равенств с неизвестными комплексными амплитудами и частотой колебаний:

$$\omega^2 X_1 = \omega_0^2(1+\kappa)X_1 - \kappa\omega_0^2 X_2,$$
$$\omega^2 X_2 = -\kappa\omega_0^2 X_1 + \omega_0^2(1+\kappa)X_2.$$

В векторно-матричной форме эти равенства можно записать как

$$\begin{pmatrix} 1+\kappa & -\kappa \\ -\kappa & 1+\kappa \end{pmatrix}\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} = \lambda\begin{pmatrix} X_1 \\ X_2 \end{pmatrix}. \qquad (3.3)$$

В этой записи использовано обозначение λ для отношения квадратов частот:

$$\lambda = \frac{\omega^2}{\omega_0^2}.$$

Возможно, равенства (3.3) по виду и напоминают СЛАУ относительно комплексных амплитуд X_1 и X_2, но таковой они не являются. Дело в том, что в правую часть (3.3) входит неизвестное число λ, которое также подлежит определению. Итак, мы пришли к необходимости решения задачи о нахождении таких чисел и векторов, для которых умножение вектора на матрицу (слева) эквивалентно умножению этого вектора на число. Такая задача называется задачей на собственные значения матрицы или матричной проблемой собственных значений (МПСЗ).

Стандартная запись МПСЗ имеет вид

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}, \qquad (3.4)$$

а терминология такова: λ – собственное значение квадратной матрицы $\mathbf{A}$, $\mathbf{x}$ – собственный вектор, соответствующий этому собственному значению. Количество собственных значений и собственных векторов у матрицы $\mathbf{A}$ ещё предстоит установить.

Несколько забегая вперёд, приведем решение в МПСЗ для колебаний связанных пружинных маятников (3.1). Будем использовать математический пакет Mathcad, в котором есть встроенная функция *eigenvals(A)*, возвращающая вектор с компонентами из собственных значений матрицы. Для вычисления собственных векторов пакет предлагает использовать встроенные функции *eigenvec(A,λ)* или *eigenvecs(A)*. Вычисления в аналитическом режиме проведём по программе (Pr.4).

$$A := \begin{pmatrix} 1+\kappa & -\kappa \\ -\kappa & 1+\kappa \end{pmatrix} \quad \lambda := \text{eigenvals}(A) \quad \lambda \to \begin{pmatrix} 1 \\ 1+2\cdot\kappa \end{pmatrix}$$

$$x := \text{eigenvecs}(A) \quad x \to \begin{pmatrix} \frac{1}{2}\cdot\sqrt{2} & \frac{-1}{2}\cdot\sqrt{2} \\ \frac{1}{2}\cdot\sqrt{2} & \frac{1}{2}\cdot\sqrt{2} \end{pmatrix} \quad x^{T}\cdot x \to \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \text{(Pr.4)}$$

$$x^{\langle 0\rangle T}\cdot x^{\langle 1\rangle} \to 0 \quad x^{\langle 0\rangle T}\cdot x^{\langle 0\rangle} \to 1 \quad x^{\langle 1\rangle T}\cdot x^{\langle 1\rangle} \to 1$$

Основные комментарии к результатам решения МПСЗ (3.3) состоят в следующем. Найдено два собственных значения матрицы 2×2 (первая строка программы). Эти значения определяют две возможных частоты свободных колебаний в системе – две нормальных частоты.

$$\omega_1 = \omega_0 \text{ и } \omega_2 = \frac{\omega_0}{\sqrt{1+2\kappa}}.$$

Первой из них соответствует собственный вектор $\mathbf{x}_1 = \left(1/\sqrt{2}, 1/\sqrt{2}\right)^T$, второй – вектор $\mathbf{x}_2 = \left(-1/\sqrt{2}, 1/\sqrt{2}\right)^T$ (вторая строка программы). Эти векторы образуют ортонормированный базис (третья строка программы). Нетрудно заметить, что вектор $\mathbf{x}_1$ описывает синфазные колебания масс. При этом пружина связи не деформируется и частота колебаний остаётся равной собственной частоте ω_0 невозмущённого осциллятора.

3.2. Характеристический полином и количество собственных значений

Дополнительные сведения о собственных значениях невырожденной квадратной матрицы $\mathbf{A}$ размером $N\times N$ можно получить, переписав задачу (3.4) следующим образом

$$(\mathbf{A}-\lambda\mathbf{I})\mathbf{x} = 0. \qquad (3.5)$$

Здесь $\mathbf{I}$ – единичная матрица. Равенства (3.5) формально представляют собой однородную СЛАУ, в отношении которой известно, что ненулевой вектор решения $\mathbf{x}$ существует лишь при условии равенства нулю определителя матрицы $\mathbf{A}-\lambda\mathbf{I}$:

$$Det(\mathbf{A}-\lambda\mathbf{I}) = 0. \qquad (3.6)$$

Это уравнение носит название характеристического уравнения матрицы $\mathbf{A}$.

Нетрудно понять, что левая часть уравнения (3.6) является полиномом степени N – характеристическим полиномом. Таким образом, собственные

значения матрицы являются корнями её характеристического полинома $P_N(\lambda) = Det(\mathbf{A} - \lambda\mathbf{I})$. При этом известно, что полином степени N имеет ровно N корней (с учётом кратности). Следовательно, в отношении матрицы справедливо утверждение о том, что невырожденная квадратная матрица $\mathbf{A}$ размером $N \times N$ имеет N собственных значений, некоторые из которых могут быть кратными. Основные свойства собственных значений перечислены в приложении D.

Полная проблема собственных значений – это решение задачи о нахождении всех собственных значений λ и всех собственных векторов $\mathbf{x}$. В противоположность этому, если требуется отыскать лишь одно или несколько λ и $\mathbf{x}$, речь идет о частичной МПСЗ.

В любом случае решение уравнения $P_N(\lambda) = 0$ не следует рассматривать как практический способ вычисления собственных значений матрицы. Причина этого состоит в том, что не существует точных формул нахождения корней полиномов степени $N \geq 5$ (теорема Галуа). Следовательно, корни придётся искать численно. Но, как было показано ранее, вычисление определителя матрицы – очень неэффективная процедура. Поэтому для практических вычислений следует использовать итерационные алгоритмы, не связанные с определителями.

3.3. Методы решения частичной проблемы собственных значений

3.3.1. Степенной метод

Основное содержание метода составляет итерационный процесс преобразования вектора начального приближения в вектор приблизительно коллинеарный собственному вектору матрицы, отвечающему её наибольшему по модулю собственному значению. Рассмотрим теоретические предпосылки для организации итераций.

Пусть квадратная матрица $\mathbf{A}$ размером $N \times N$ имеет N различных собственных значений, расположенных в порядке возрастания модулей, а соответствующие им собственные векторы $\mathbf{x}_n$ образуют ортонормированный базис. Тогда произвольный вектор начального приближения можно представить в виде разложения

$$\mathbf{u}_0 = \sum_{n=1}^{N} c_n \mathbf{x}_n . \qquad (3.7)$$

Новый вектор $\mathbf{u}_1$ создадим путём преобразования вектора $\mathbf{u}_0$

$$\mathbf{u}_1 = \mathbf{A}\mathbf{u}_0$$

и разложим его по базису собственных векторов

$$\mathbf{u}_1 = \sum_{n=1}^{N-1} \lambda_n c_n \mathbf{x}_n + \lambda_N c_N \mathbf{x}_N . \qquad (3.8)$$

Так как $|\lambda_N| > |\lambda_{N-1}| > ... > |\lambda_1|$, то в разложении (3.8) по сравнению с разложением (3.7) вклад отдельно выделенного последнего слагаемого начинает преобладать над вкладами остальных слагаемых. Это преобладание усиливается в итерационном процессе

$$\mathbf{u}_2 = \mathbf{A}\mathbf{u}_1,\ \mathbf{u}_3 = \mathbf{A}\mathbf{u}_2,\ ...,\ \mathbf{u}_k = \mathbf{A}\mathbf{u}_{k-1}, ...$$

В результате *k*-ой итерации формируется вектор

$$\mathbf{u}_k = \lambda_N^k \left(\sum_{n=1}^{N-1} \left(\frac{\lambda_n}{\lambda_N} \right)^k c_n \mathbf{x}_n + c_N \mathbf{x}_N \right).$$

Понятно, что при $k \to \infty$

$$\frac{\mathbf{u}_k}{\lambda_N^k} \to c_N \mathbf{x}_N , \qquad (3.9)$$

причём скорость сходимости тем выше, чем меньше отношение $|\lambda_{N-1}| / |\lambda_N|$. Для нормированного вектора предел (3.9) выглядит как

$$\frac{\mathbf{u}_k}{\|\mathbf{u}_k\|} \to \frac{c_N \lambda_N^k}{\left| c_N \lambda_N^k \right|} \mathbf{x}_N .$$

Таким образом, в процессе итераций формируется вектор, параллельный собственному вектору матрицы.

Согласно (3.9), с ростом номера итерации все более точным становится приближённое равенство

$$\mathbf{u}_k \approx \lambda_N \mathbf{u}_{k-1} .$$

Следовательно, любую из проекций векторов $\mathbf{u}_k$ и $\mathbf{u}_{k-1}$ можно использовать для вычисления *k*-го приближения собственного значения λ_N:

$$\lambda_N^{(k)} = \frac{(\mathbf{u}_k)_i}{(\mathbf{u}_{k-1})_i} .$$

Отметим также, что на основе существования предела (3.9) *k*-ое приближение можно вычислять с помощью отношения Рэлея:

$$\lambda_N^{(k)} = \frac{(\mathbf{u}_k, \mathbf{A}\mathbf{u}_k)}{(\mathbf{u}_k, \mathbf{u}_k)}, \qquad (3.10)$$

где $(\mathbf{a},\mathbf{b}) = \mathbf{a}^T \mathbf{b}$ – скалярное произведение векторов.

Ниже приведена функция Mathcad *EGmax(A,u0,ε)*, реализующая вычисления максимального собственного значения и соответствующего ему собственного вектора степенным методом.

$$
\begin{aligned}
&\mathrm{EGmax}(A, u0, \varepsilon) := \\
&\quad \text{Поиск\_наибольшего\_СЗ\_матрицы} \leftarrow \% \\
&\quad \text{степенным\_методом} \leftarrow \% \\
&\quad k \leftarrow 1 \\
&\quad \lambda_0 \leftarrow 1 \\
&\quad \text{while } 1 \\
&\qquad u \leftarrow A \cdot u0 \\
&\qquad \lambda_k \leftarrow \frac{\left(u0^{T} \cdot u\right)_0}{\left(u0^{T} \cdot u0\right)_0} \\
&\qquad u0 \leftarrow \frac{u}{|u|} \\
&\qquad \text{return } \mathrm{stack}(\lambda_k, u0) \text{ if } |\lambda_k - \lambda_{k-1}| \le \varepsilon \\
&\qquad k \leftarrow k + 1
\end{aligned}
$$

В программе на каждой итерации производится нормализация (деление на модуль) вектора $\mathbf{u}_k$. Эта операция позволяет получить на выходе вектор единичной длины и предотвратить переполнение или исчезновение порядка в итерационном процессе.

3.3.2. Обратный степенной метод

Итерационный алгоритм степенного метода можно изменить, так чтобы он позволял вычислить наименьшее по модулю собственное значение матрицы и соответствующий этому значению собственный вектор. Для обоснования изменений рассмотрим стандартную МПСЗ матрицы $\mathbf{A}$

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}.$$

Умножим слева это равенство на обратную матрицу $\mathbf{A}^{-1}$. Получим $\mathbf{x} = \lambda\mathbf{A}^{-1}\mathbf{x}$ или

$$\mathbf{A}^{-1}\mathbf{x} = \frac{1}{\lambda}\mathbf{x},$$

а это не что иное, как МПСЗ матрицы $\mathbf{A}^{-1}$. Таким образом, при обращении матрицы её собственные векторы сохраняются, а собственные значения заменяются на обратные

$$\lambda\left(\mathbf{A}^{-1}\right)=\lambda^{-1}(\mathbf{A}).$$

Установленное свойство собственных значений позволяет организовать следующий итерационный процесс преобразования вектора начального приближения $\mathbf{v}_0$ в собственный вектор матрицы:

$$\mathbf{v}_1=\mathbf{A}^{-1}\mathbf{v}_0,\ \mathbf{v}_2=\mathbf{A}^{-1}\mathbf{v}_1,\ \ldots,\ \mathbf{v}_k=\mathbf{A}^{-1}\mathbf{v}_{k-1},\ldots$$

Считается, что, т.к. обращение матрицы требует большего числа арифметических операций, чем решение соответствующей СЛАУ, то преобразования вектора на каждой итерации следует проводить не в прямой форме, а как решение системы

$$\mathbf{A}\,\mathbf{v}_k=\mathbf{v}_{k-1}.$$

При этом можно один раз провести *LU*-разложение матрицы и выполнить несколько обратных подстановок (по числу итераций, необходимых для достижения заданной точности). В приведенной ниже демонстрационной Mathcad-функции *EGmin(A,v0,ε)* используется именно такой способ преобразований векторов, правда, *LU*-разложение не применяется. Приближение для собственного значения вычисляется на основе обратного отношения Рэлея (сравните с (3.10)):

$$\lambda_N^{(k)}=\frac{\left(\mathbf{v}_k,\mathbf{v}_k\right)}{\left(\mathbf{v}_k,\mathbf{A}\mathbf{v}_k\right)}.$$

$\mathrm{EGmin}(A, v0, \varepsilon) :=$ | Поиск_наименьшего_СЗ_матрицы ← %
| обратным_степенным_методом ← %
| $k \leftarrow 1$
| $\lambda_0 \leftarrow 1$
| while 1
| | $v \leftarrow \mathrm{lsolve}(A, v0)$
| | $\lambda_k \leftarrow \dfrac{\left(v0^{T}\cdot v0\right)_0}{\left(v0^{T}\cdot v\right)_0}$
| | $v0 \leftarrow \dfrac{v}{\lvert v\rvert}$
| | return $\mathrm{stack}(\lambda_k, v0)$ if $\lvert\lambda_k-\lambda_{k-1}\rvert\le\varepsilon$
| | $k \leftarrow k+1$

3.3.3. Обратный степенной метод со смещением

Итерационный алгоритм для обратной матрицы позволяет найти произвольное собственное значение, если применять его к трансформированной определенным способом исходной матрице $\mathbf{A}$. Способ трансформации легко обосновать следующими рассуждениями.

Рассмотрим матрицу

$$\mathbf{B} = \mathbf{A} - \alpha\mathbf{I},$$

где α – произвольная константа, а $\mathbf{I}$ – единичная матрица. МПСЗ матрицы $\mathbf{B}$ имеет вид

$$(\mathbf{A} - \alpha\mathbf{I})\mathbf{x} = \lambda(\mathbf{B})\mathbf{x}$$

и легко преобразуется к форме

$$\mathbf{A}\mathbf{x} = (\lambda(\mathbf{B}) + \alpha)\mathbf{x}.$$

Из этой записи следует, что

$$\lambda(\mathbf{A}) = \lambda(\mathbf{B}) + \alpha \text{ или } \lambda(\mathbf{B}) = \lambda(\mathbf{A}) - \alpha.$$

Таким образом, выбрав константу α в окрестности некоторого собственного значения λ_m, можно найти его обратным степенным методом, применённым к матрице $\mathbf{B}$.

```
EG(A, w0, α, ε) := | Поиск_произвольного_СЗ_матрицы ← %
                   | обратным_степенным_методо_со_смещением ← %
                   | N ← length(w0)
                   | I ← identity(N)
                   | B ← A − α·I
                   | k ← 1
                   | λ_0 ← 1
                   | while 1
                   |     w ← lsolve(B, w0)
                   |     λ_k ← (w0^T · w0)_0 / (w0^T · w)_0
                   |     w0 ← w / |w|
                   |     return stack(λ_k + α, w0) if |λ_k − λ_{k−1}| ≤ ε
                   |     k ← k + 1
```

Обратный степенной метод со смещением реализуется Mathcad-функцией *EG(A,w0,α,ε)*. Метод следует применять, если области локализации собственных значений $\lambda(\mathbf{A})$ известны заранее.

3.4. QR-метод решения полной проблемы собственных значений

Одним из наиболее широко используемых методов решения полной проблемы собственных значений является так называемый *QR*-метод (*QR*-алгоритм). В основе метода лежит представление матрицы **A** в виде произведения двух матриц (*QR*-разложения):

$$\mathbf{A} = \mathbf{QR},$$

одна из которых **Q** – ортогональная матрица, а другая **R** – верхняя треугольная. Напомним, что матрица называется ортогональной, если ее обратная матрица равна транспонированной: $\mathbf{Q}^{-1} = \mathbf{Q}^{T}$.

Итерационный процесс *QR*-метода строится путем многократного повторения процедуры *QR*-разложения первичной матрицы и процедуры *RQ*-произведения для формирования вторичной матрицы. *QR*-алгоритм выглядит следующим образом:

итерация 1:

$\mathbf{A} = \mathbf{Q}^{(0)}\mathbf{R}^{(0)}$ – *QR*-разложение,

$\mathbf{A}^{(1)} = \mathbf{R}^{(0)}\mathbf{Q}^{(0)}$ – *RQ*-произведение;

итерация 2:

$\mathbf{A}^{(1)} = \mathbf{Q}^{(1)}\mathbf{R}^{(1)}$ – *QR*-разложение,

$\mathbf{A}^{(2)} = \mathbf{R}^{(1)}\mathbf{Q}^{(1)}$ – *RQ*-произведение;

……

итерация k:

$\mathbf{A}^{(k-1)} = \mathbf{Q}^{(k-1)}\mathbf{R}^{(k-1)}$ – *QR*-разложение,

$\mathbf{A}^{(k)} = \mathbf{R}^{(k-1)}\mathbf{Q}^{(k-1)}$ – *RQ*-произведение.

В результате процесса образуется последовательность матриц $\mathbf{A}^{(1)}$, $\mathbf{A}^{(2)}$, …, $\mathbf{A}^{(k)}$. Нетрудно показать, что все они подобны и, следовательно, имеют одинаковые собственные значения. Действительно, выполнив преобразования

$$\mathbf{A}^{(k)} = \mathbf{R}^{(k-1)}\mathbf{Q}^{(k-1)} = \left(\mathbf{Q}^{(k-1)}\right)^{T}\mathbf{Q}^{(k-1)}\mathbf{R}^{(k-1)}\mathbf{Q}^{(k-1)} = \left(\mathbf{Q}^{(k-1)}\right)^{-1}\mathbf{A}^{(k-1)}\mathbf{Q}^{(k-1)},$$

мы убеждаемся в подобии матриц $\mathbf{A}^{(k)}$ и $\mathbf{A}^{(k-1)}$ для любого *k*.

Замечательным свойством последовательности $\left\{\mathbf{A}^{(k)}\right\}$ является то, что при отсутствии у исходной матрицы **A** одинаковых по модулю собственных значений она сходится к верхней треугольной матрице $\mathbf{R}^{(\infty)}$:

$$\lim_{k\to\infty} \mathbf{A}^{(k)} = \mathbf{R}^{(\infty)}.$$

Если **A** симметрична, то $\mathbf{R}^{(\infty)}$ трёхдиагональна. Все собственные значения матрицы **A** расположены на главной диагонали матрицы $\mathbf{R}^{(\infty)}$.

В качестве критерия окончания итерационного процесса можно использовать условие малости величин поддиагональных элементов $a_{n+1,n}^{(k)}$ матрицы $\mathbf{A}^{(k)}$ или неравенство

$$\sum_{n=1}^{N} \left| a_{n,n}^{(k)} - a_{n,n}^{(k-1)} \right| < \varepsilon .$$

Ниже приведен текст функции Mathcad *QREig(A,ε)*, реализующей *QR*-метод для матриц с различными вещественными собственными значениями.

```
QREig(A, ε) := | QR_метод_для_вещественных_СЗ ← %
               | M ← rows(A)
               | while 1
               |     A0 ← A
               |     QR ← qr(A)
               |     Q ← submatrix(QR, 0, M − 1, 0, M − 1)
               |     R ← submatrix(QR, 0, M − 1, M, 2·M − 1)
               |     A ← R·Q
               |     S ← 0
               |     for m ∈ 0.. M − 1
               |         λ_m ← A_m,m
               |         S ← S + (A_m,m − A0_m,m)^2
               |     return λ if √S ≤ ε
```

Вычислительная эффективность *QR*-алгоритма определяется, главным образом, эффективностью *QR*-разложения. Если **A** – матрица произвольного вида, то разложение на каждой итерации требует порядка N^3 операций. Для повышения эффективности *QR*-разложения сначала исходная матрица преобразованием подобия трансформируется к форме Хессенберга – верхней треугольной форме с дополнительной нижней поддиагональю. У симметричных матриц форма Хессенберга является трёхдиагональной. *QR*-разложение матрицы формы Хессенберга можно выполнить за N^2 операций.

Причём очень важно, что в процессе итераций все последующие матрицы $\mathbf{A}^{(k)}$ сохраняют форму Хессенберга.

Приведённая программа *QREig(A,ε)* использует встроенную в среду Mathcad-функцию *QR*-разложения $qr(A)$.

3.5. МПСЗ и корни полиномов

Ранее было установлено, что не следует искать собственные значения матрицы как корни её характеристического полинома. Но, если мы имеем эффективный метод решения МПСЗ, то можно решать обратную задачу – искать корни полинома как собственные значения матрицы, для которой этот полином является характеристическим.

Пусть задан полином достаточно общего вида

$$P_N(\lambda) = \lambda^N + b_{N-1}\lambda^{N-1} + \ldots + b_1\lambda + b_0. \qquad (3.11)$$

Нетрудно установить, что он является характеристическим для матрицы

$$\mathbf{B} = \begin{pmatrix} 0 & 0 & \ldots & 0 & -b_0 \\ 1 & 0 & \ldots & 0 & -b_1 \\ 0 & 1 & \ldots & 0 & -b_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 1 & -b_{N-1} \end{pmatrix}.$$

Следовательно, совокупность собственных значений $\lambda(\mathbf{B})$ одновременно составляет совокупность корней полинома (3.11). Вычисления можно провести, воспользовавшись Mathcad-функцией *PolRootEig(b)*.

```
PolRootEig(b) := | Поиск_корней_полинома ← %
                 | методом_решения_МПСЗ ← %
                 | N ← length(b) – 1
                 | for n ∈ 0.. N – 1
                 |     for m ∈ 0.. N – 1
                 |       B[n,m] ← 0
                 |     B[n,N–1] ← –b[n]
                 | for n ∈ 1.. N – 1
                 |   B[n,n–1] ← 1
                 | eigenvals(B)
```

3.6. Обобщённая задача на собственные значения

Многие физические приложения сводятся к решению более общей, чем (3.4), задачи:

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{B}\mathbf{x}, \tag{3.12}$$

где **A** и **B** – квадратные матрицы, **x** – вектор-столбец, λ – скалярная величина. Задача (3.12) называется обобщённой задачей на собственные значения или обобщённой матричной проблемой собственных значений.

Рассмотрим, например, свободные колебания в электрической схеме, изображённой на рис. 3.2 и представляющей собой два *LC*-контура, связанных общим магнитным полем (коэффициент взаимоиндукции *M* – характеристика связи).

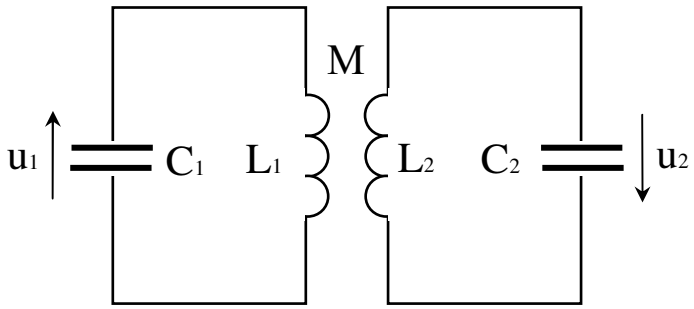

Рис. 3.2. Колебательная система

Применяя законы Кирхгофа, нетрудно сформировать математическую модель рассматриваемой схемы в виде системы обыкновенных дифференциальных уравнений относительно напряжений $u_1(t)$ и $u_2(t)$ на ёмкостях C_1 и C_2

$$\begin{aligned} \frac{d^2u_1}{dt^2} + \omega_1^2 u_1 &= -k_1 \frac{d^2u_2}{dt^2}, \\ \frac{d^2u_2}{dt^2} + \omega_2^2 u_2 &= -k_2 \frac{d^2u_1}{dt^2}. \end{aligned} \tag{3.13}$$

Здесь $\omega_1 = 1/\sqrt{L_1C_1}$ и $\omega_2 = 1/\sqrt{L_2C_2}$ – собственные частоты одиночных контуров, $k_1 = M/L_1$ и $k_2 = M/L_2$ – коэффициенты связи.

Решение системы (3.13) ищем в форме гармонических колебаний

$$u_1(t) = x_1 \exp(j\omega t),\ u_2(t) = x_2 \exp(j\omega t).$$

После дифференцирования и сокращения на экспоненту система дифференциальных уравнений (3.13) преобразуется в систему алгебраических равенств с неизвестными комплексными амплитудами x_1, x_2 и частотой колебаний:

$$-\omega^2 x_1 + \omega_1^2 x_1 = k_1 \omega^2 x_2,$$
$$-\omega^2 x_2 + \omega_2^2 x_2 = k_2 \omega^2 x_1.$$

В векторно-матричной форме эти равенства можно записать как

$$\begin{pmatrix} \omega_1^2 & 0 \\ 0 & \omega_2^2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \omega^2 \begin{pmatrix} 1 & k_1 \\ k_2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}. \tag{3.14}$$

Для матриц

$$\mathbf{A} = \begin{pmatrix} \omega_1^2 & 0 \\ 0 & \omega_2^2 \end{pmatrix}, \ \mathbf{B} = \begin{pmatrix} 1 & k_1 \\ k_2 & 1 \end{pmatrix},$$

вектора $\mathbf{x} = (x_1, x_2)^T$ и числа $\lambda = \omega^2$ выражение (3.14) даёт нам стандартную запись обобщённой МПСЗ.

Если матрицы **A** и **B** невырождены (или этим свойством обладает хотя бы одна из них), то обобщённую МПСЗ можно свести к стандартной. Действительно, умножив (3.12) слева на обратную матрицу $\mathbf{B}^{-1}$, получим

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{x} = \lambda \mathbf{x},$$

т.е. стандартную задачу на собственные значения матрицы $\mathbf{D} = \mathbf{B}^{-1}\mathbf{A}$:

$$\mathbf{D}\mathbf{x} = \lambda \mathbf{x}, \tag{3.15}$$

Следовательно, собственные значения $\lambda(\mathbf{D})$ будут также и собственными значениями $\lambda(\mathbf{A},\mathbf{B})$ задачи (3.12). Совпадают и собственные векторы задач (3.12) и (3.15).

Ещё одна возможность сведения обобщённой МПСЗ к стандартной состоит в умножении (3.12) слева на обратную матрицу $\mathbf{A}^{-1}$ и преобразовании результата умножения к виду

$$\mathbf{G}\mathbf{x} = \frac{1}{\lambda}\mathbf{x}, \tag{3.16}$$

где $\mathbf{G} = \mathbf{A}^{-1}\mathbf{B}$. Отсюда следует, что собственные значения $\lambda(\mathbf{G})$ матрицы **G** и собственные значения $\lambda(\mathbf{A},\mathbf{B})$ обобщённой задачи (3.12) связаны соотношением

$$\lambda(\mathbf{A},\mathbf{B}) = \frac{1}{\lambda(\mathbf{G})}.$$

С учётом этой связи совпадают также и соответствующие собственные векторы задач (3.12) и (3.16).

Путём преобразований, аналогичных проведённым выше, нетрудно показать, что ещё две стандартные МПСЗ

$$\mathbf{A}\mathbf{B}^{-1}\mathbf{y} = \lambda\mathbf{y} \text{ и } \mathbf{B}\mathbf{A}^{-1}\mathbf{y} = \frac{1}{\lambda}\mathbf{y}$$

позволяют вычислить собственные значения обобщённой задачи (3.12):

$$\lambda(\mathbf{A},\mathbf{B}) = \lambda(\mathbf{A}\mathbf{B}^{-1}) \text{ и } \lambda(\mathbf{A},\mathbf{B}) = \frac{1}{\lambda(\mathbf{B}\mathbf{A}^{-1})}.$$

В справедливости полученных связей можно убедиться вычислениями по программе (Pr.5).

$$\begin{aligned}
&A := \begin{pmatrix} 4 & 1 & 2 \\ 2 & 6 & 3 \\ 3 & 1 & 7 \end{pmatrix} \quad B := \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \quad \text{genvals}(A,B) = \begin{pmatrix} 16.27 \\ 1.178 \\ 1.553 \end{pmatrix} \\
&\text{eigenvals}\left(B^{-1}\cdot A\right) = \begin{pmatrix} 16.27 \\ 1.178 \\ 1.553 \end{pmatrix} \qquad \frac{1}{\text{eigenvals}\left(A^{-1}\cdot B\right)} = \begin{pmatrix} 1.178 \\ 16.27 \\ 1.553 \end{pmatrix} \\
&\text{eigenvals}\left(A\cdot B^{-1}\right) = \begin{pmatrix} 16.27 \\ 1.178 \\ 1.553 \end{pmatrix} \qquad \frac{1}{\text{eigenvals}\left(B\cdot A^{-1}\right)} = \begin{pmatrix} 1.178 \\ 16.27 \\ 1.553 \end{pmatrix}
\end{aligned} \quad \text{(Pr.5)}$$

Несмотря на теоретическую правомерность сведения обобщённой МПСЗ к стандартной, на практике такой подход может быть сопряжён со значительными вычислительными погрешностями. Например, если матрица **B** плохо обусловлена, то решения стандартной МПСЗ с матрицами $\mathbf{B}^{-1}\mathbf{A}$ или $\mathbf{A}\mathbf{B}^{-1}$ могут быть слишком неточными решениями обобщённой задачи (3.12). Кроме того, при умножении на обратную матрицу часто теряются полезные свойства исходных матриц **A** и **B**, такие как симметрия или разреженность. Поэтому разработаны и часто применяются алгоритмы прямого решения обобщённой МПСЗ. В математическом пакете Mathcad для вычисления обобщённых собственных значений предназначена встроенная функция *genvals(A,B)*, а обобщённые собственные векторы вычисляются с помощью функции *genvecs(A,B)*.

Программа (Pr.6) в режиме аналитических преобразований решает задачу (3.14) о расчёте частот собственных колебаний двух идентичных индуктивно связанных *LC*-контуров. Краткий комментарий к решению состоит в следующем.

$$A := \begin{bmatrix} (\omega_1)^2 & 0 \\ 0 & (\omega_1)^2 \end{bmatrix} \qquad \text{genvals}(A, B) \rightarrow \begin{bmatrix} \frac{-(\omega_1)^2}{(k-1)} \\ \frac{(\omega_1)^2}{(k+1)} \end{bmatrix}$$

$$B \equiv \begin{pmatrix} 1 & k \\ k & 1 \end{pmatrix} \qquad \text{eigenvecs}\left(A^{-1} \cdot B\right) \rightarrow \begin{pmatrix} \frac{1}{2}\cdot\sqrt{2} & \frac{-1}{2}\cdot\sqrt{2} \\ \frac{1}{2}\cdot\sqrt{2} & \frac{1}{2}\cdot\sqrt{2} \end{pmatrix} \qquad \text{(Pr.6)}$$

Наличие связи между колебательными системами приводит к расщеплению частот связанных колебаний – вместо собственных колебаний независимых осцилляторов с частотой ω_1 в связанной системе могут существовать двухчастотные колебания с частотами

$$\omega_1^{(c)} = \frac{\omega_1}{\sqrt{1-k}} \text{ и } \omega_2^{(c)} = \frac{\omega_1}{\sqrt{1+k}}.$$

Высшая частота $\omega_1^{(c)}$ соответствует синфазным колебаниям с собственным вектором $\mathbf{x}_1 = \left(1/\sqrt{2}, 1/\sqrt{2}\right)^T$, низшая $\omega_2^{(c)}$ – противофазным колебаниям с собственным вектором $\mathbf{x}_2 = \left(-1/\sqrt{2}, 1/\sqrt{2}\right)^T$. Это полностью соответствует физическим представлениям о свойствах схемы, представленной на рис. 3.2.

Приложение А. Нормы векторов и матриц

Полностью векторы и матрицы характеризуются совокупностью своих элементов. Вместе с тем, часто полезно иметь одно число, указывающее на величину вектора или матрицы. Для вектора таким числом может служить его длина (модуль). Но используются и другие числа – нормы вектора.

Норма $\|\mathbf{x}\|$ вектора $\mathbf{x}$ – это число со следующими свойствами:

$$\text{а) } \|\mathbf{x}\| \geq 0, \text{ причем } \|\mathbf{x}\| = 0, \text{ если } \mathbf{x} = 0;$$

$$\text{б) } \|\alpha\mathbf{x}\| = \alpha\|\mathbf{x}\|, \text{ где } \alpha \text{ – скаляр;}$$

$$\text{в) } \|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|.$$

Наиболее распространенные нормы:

$$\|\mathbf{x}\|_1 = \sum_{n=1}^{N} |x_n|, \tag{A.1}$$

$$\|\mathbf{x}\|_e = \sqrt{\sum_{n=1}^{N} x_n^2} = |\mathbf{x}|, \tag{A.2}$$

$$\|\mathbf{x}\|_i = \max(|x_n|). \tag{A.3}$$

Они носят названия: (А.1) – первая норма, (А.2) – евклидова норма, (А.3) – равномерная норма.

Любую из указанных норм формально можно использовать для измерения длины вектора. Другими словами, норма $\|\mathbf{x}\|$ является обобщенной длиной вектора $\mathbf{x}$.

Норма $\|\mathbf{A}\|$ квадратной матрицы $\mathbf{A}$ – это число, обладающее свойствами:

$$\text{а) } \|\mathbf{A}\| \geq 0, \text{ причем } \|\mathbf{A}\| = 0, \text{ если } \mathbf{A} = 0;$$

$$\text{б) } \|\alpha\mathbf{A}\| = \alpha\|\mathbf{A}\|, \text{ где } \alpha \text{ – скаляр;}$$

$$\text{в) } \|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|,$$

$$\text{г) } \|\mathbf{A} \cdot \mathbf{B}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{B}\|.$$

Последнее из этих свойств позволяет согласовать норму матрицы с нормой вектора: для согласованных векторно-матричных норм имеет место неравенство

$$\|\mathbf{A} \cdot \mathbf{x}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{x}\|.$$

Согласованными с векторными нормами (А.1) – (А.3) являются соответственно первая, евклидова и равномерная нормы матриц:

$$\|\mathbf{A}\|_1 = \max_m \left(\sum_{n=1}^{N} |A_{nm}| \right), \tag{A.4}$$

$$\|\mathbf{A}\|_e = \sqrt{\sum_{n=1}^{N} \sum_{m=1}^{N} A_{nm}^2}\ , \tag{A.5}$$

$$\|\mathbf{A}\|_i = \max_n \left(\sum_{m=1}^{N} |A_{nm}| \right). \tag{A.6}$$

При этом норма $\|\mathbf{A}\|_i$ согласована со всеми нормами (A.1) – (A.3).

Кроме перечисленных, применение находит также вторая норма

$$\|\mathbf{A}\|_2 = \sqrt{\rho\left(\mathbf{A}^T \mathbf{A}\right)},$$

где $\rho\left(\mathbf{A}^T \mathbf{A}\right)$ – спектральный радиус матрицы – максимальное собственное значение матрицы $\mathbf{A}^T \mathbf{A}$.

В среде Mathcad вычисления норм матриц реализуются с помощью встроенных функций: *norm1(A)*, *norme(A)*, *normi(A)* и *norm2(A)*. Mathcad-программа (Pr.A) служит примером вычислений.

$$\text{NORM1}(A) := \left| \begin{array}{l} N \leftarrow \text{rows}(A) \\ \text{for } m \in 0..N-1 \\ \quad S_m \leftarrow \sum_{n=0}^{N-1} |A_{n,m}| \\ \max(S) \end{array} \right. \qquad \text{NORMi}(A) := \left| \begin{array}{l} N \leftarrow \text{rows}(A) \\ \text{for } n \in 0..N-1 \\ \quad S_n \leftarrow \sum_{m=0}^{N-1} |A_{n,m}| \\ \max(S) \end{array} \right.$$

$$\text{NORME}(A) := \left| \begin{array}{l} N \leftarrow \text{rows}(A) \\ \sqrt{\sum_{m=0}^{N-1} \sum_{n=0}^{N-1} \left(A_{n,m}\right)^2} \end{array} \right. \qquad \text{NORM2}(A) := \left| \begin{array}{l} N \leftarrow \text{rows}(A) \\ \lambda \leftarrow \text{eigenvals}\left(A^T \cdot A\right) \\ \sqrt{\max(\lambda)} \end{array} \right. \tag{Pr.A}$$

$$A \equiv \begin{pmatrix} 3 & 8 & 9 \\ 1 & 5 & 2 \\ 7 & 4 & 6 \end{pmatrix}$$

NORM1(A) = 17 norm1(A) = 17

NORMi(A) = 20 normi(A) = 20

NORME(A) = 16.882 norme(A) = 16.882

NORM2(A) = 16.134 norm2(A) = 16.134

Приложение В. Погрешности решения СЛАУ

1. Пусть вектор правых частей системы уравнений

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{В.1}$$

получает приращение $\boldsymbol{\beta}$. При этом вектор решения получит приращение $\boldsymbol{\xi}$ и возмущенная СЛАУ будет выглядеть следующим образом:

$$\mathbf{A}(\mathbf{x} + \boldsymbol{\xi}) = \mathbf{b} + \boldsymbol{\beta}. \tag{В.2}$$

Из системы (В.2) с учётом (В.1) находим

$$\boldsymbol{\xi} = \mathbf{A}^{-1}\boldsymbol{\beta}, \tag{В.3}$$

где $\mathbf{A}^{-1}$ – обратная матица.

Воспользуемся неравенством, справедливым для согласованных векторно-матричных норм:

$$\|\mathbf{AB}\| \le \|\mathbf{A}\|\|\mathbf{B}\|.$$

Для нормы приращения вектора решения получим

$$\|\boldsymbol{\xi}\| \le \|\mathbf{A}^{-1}\|\|\boldsymbol{\beta}\|, \tag{В.4}$$

а из системы (В.1) находим

$$\|\mathbf{x}\| \ge \frac{\|\mathbf{b}\|}{\|\mathbf{A}\|}. \tag{В.5}$$

Теперь на основе неравенств (В.4) и (В.5) нетрудно получить неравенство, ограничивающее относительное приращение нормы вектора решения СЛАУ $\|\delta\mathbf{x}\| = \|\boldsymbol{\xi}\|/\|\mathbf{x}\|$, вызванное изменением вектора правых частей с относительной нормой $\|\delta\mathbf{b}\| = \|\boldsymbol{\beta}\|/\|\mathbf{b}\|$:

$$\|\delta\mathbf{x}\| \le \|\mathbf{A}\|\|\mathbf{A}^{-1}\|\|\delta\mathbf{b}\|. \tag{В.6}$$

Как и следовало ожидать, относительное приращение $\|\delta\mathbf{x}\|$ ограничено сверху, положение этой границы существенным образом зависит от скалярной величины

$$cond(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|, \tag{В.7}$$

называемой числом обусловленности матрицы $\mathbf{A}$.

Заметим, что приращение $\|\delta\mathbf{b}\|$ входного вектора $\mathbf{b}$ может возникнуть из-за погрешностей экспериментальных данных или погрешностей предыдущих вычислений. Тогда ограничение на погрешность решения $\|\delta\mathbf{x}\|$

полностью определяется числом обусловленности $cond(\mathbf{A})$. Если $cond(\mathbf{A}) >> 1$, то погрешность решения может быть весьма высока.

2. Рассмотрим теперь ситуацию, в которой приращения получают элементы матрицы СЛАУ (В.1) и возмущенная система выглядит следующим образом

$$(\mathbf{A} + \Delta\mathbf{A})(\mathbf{x} + \boldsymbol{\xi}) = \mathbf{b}. \qquad \text{(В.8)}$$

Вычитая из (В.8) невозмущенную систему (В.1), получим

$$(\mathbf{A} + \Delta\mathbf{A})\boldsymbol{\xi} = -\Delta\mathbf{A}\mathbf{x}.$$

Будем считать, что в этой системе произведением возмущений $\Delta\mathbf{A}\boldsymbol{\xi}$ можно пренебречь по сравнению с $\mathbf{A}\boldsymbol{\xi}$ (это стандартный прием линеаризации соотношений). Тогда СЛАУ для возмущений решения имеет вид

$$\mathbf{A}\boldsymbol{\xi} = -\Delta\mathbf{A}\mathbf{x}. \qquad \text{(В.9)}$$

Система (В.9) имеет решение

$$\boldsymbol{\xi} = -\mathbf{A}^{-1}\Delta\mathbf{A}\mathbf{x},$$

из которого следует неравенство

$$\|\boldsymbol{\xi}\| \le \|\mathbf{A}^{-1}\|\|\Delta\mathbf{A}\|\|\mathbf{x}\|.$$

Для относительного возмущения вектора решения это неравенство преобразуется к виду

$$\|\delta\mathbf{x}\| \le \|\mathbf{A}^{-1}\|\|\Delta\mathbf{A}\| = \|\mathbf{A}^{-1}\|\|\mathbf{A}\|\frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|}. \qquad \text{(В.10)}$$

Введя в рассмотрение относительную норму возмущения матрицы СЛАУ (В.1) $\|\delta\mathbf{A}\| = \|\Delta\mathbf{A}\|/\|\mathbf{A}\|$, неравенство (В.10), ограничивающее относительную норму возмущения вектора решения при возмущениях матрицы СЛАУ преобразуем к виду

$$\|\delta\mathbf{x}\| \le cond(\mathbf{A})\|\delta\mathbf{A}\|. \qquad \text{(В.11)}$$

В формулу (В.10) по-прежнему входит число обусловленности

3. Определение числа обусловленности (В.7) связано с используемой нормой матриц. Наибольшее распространение получили три сравнительно легко вычисляемых нормы: первая, евклидова и равномерная (см. Приложение А). В среде Mathcad вычисления реализуются с помощью встроенных функций. Соответственно, для характеристики матрицы используется одно из трех чисел обусловленности. Пример вычислений в Mathcad выглядит следующим образом.

$$A := \begin{pmatrix} 1 & 5 & 3 \\ 2 & 4 & 7 \\ 8 & 9 & 6 \end{pmatrix}$$

$\text{norm1}(A) = 18$	$\text{norm1}(A)\cdot\text{norm1}\left(A^{-1}\right) = 12.561$	$\text{cond1}(A) = 12.561$
$\text{norme}(A) = 16.882$	$\text{norme}(A)\cdot\text{norme}\left(A^{-1}\right) = 9.018$	$\text{conde}(A) = 9.018$
$\text{normi}(A) = 23$	$\text{normi}(A)\cdot\text{normi}\left(A^{-1}\right) = 10.755$	$\text{condi}(A) = 10.755$

Пример, в частности, показывает, что по-разному определенные числа обусловленности имеют один и тот же порядок величины.

Для числа обусловленности всегда выполняется неравенство

$$cond(A) \geq 1.$$

Если $cond(A) \sim 1 \div 10$, то погрешности входных данных слабо сказываются на решении системы (B.1) и она считается хорошо обусловленной. Если же $cond(A) > 1000$, то система плохо обусловлена. Но более точное представление об обусловленности СЛАУ должно основываться на требованиях, предъявляемых к решению. Если, например, входные погрешности имеют порядок 10^{-6}, а допустимая погрешность решения порядка 10^{-2}, то и при $cond(A) \sim 10^4$ СЛАУ еще можно считать хорошо обусловленной.

Mathcad-программа (Pr.B) вычисляет числа обусловленности матриц Вандермонда, входящих в СЛАУ для расчета интерполяционного полинома

$$V(N) := \left| \begin{array}{l} \text{for } n \in 0..N \\ \quad \left| \begin{array}{l} x_n \leftarrow \frac{1}{N}\cdot n \\ \text{for } m \in 0..N \\ \quad A_{n,m} \leftarrow (x_n)^m \end{array} \right. \\ A \end{array} \right. \qquad \text{(Pr.B)}$$

$$V(3) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0.333 & 0.111 & 0.037 \\ 1 & 0.667 & 0.444 & 0.296 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$\text{conde}(V(6)) = 3.98 \times 10^4$$

$$\text{conde}(V(8)) = 2.24 \times 10^6$$

$$\text{conde}(V(10)) = 1.299 \times 10^8$$

методом неопределенных коэффициентов.

Приложение С. Преобразование Хаусхолдера и *QR*-разложение

Преобразованием (отражением) Хаусхолдера называется операция умножения на матрицу

$$\mathbf{H} = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}, \tag{C.1}$$

где $\mathbf{v}$ – произвольный ненулевой вектор-столбец.

Нетрудно убедиться в том, что при любом ненулевом векторе $\mathbf{v}$ матрица Хаусхолдера (С.1) является симметричной ортогональной матрицей. Действительно,

$$\mathbf{H}^T = \mathbf{I}^T - 2\left(\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}\right)^T = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}} = \mathbf{H},$$

$$\mathbf{H}^T\mathbf{H} = \left(\mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}\right)^T\left(\mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}\right) = \mathbf{I} - 4\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}} + 4\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}}\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}} = \mathbf{I}.$$

При этом свобода выбора вектора $\mathbf{v}$ позволяет придать матрице $\mathbf{H}$ необходимые свойства.

Пусть преобразованием Хаусхолдера требуется обратить в нуль все компоненты вектора $\mathbf{A}_1$ за исключением первой , т.е.

$$\mathbf{R}_1 = \mathbf{H}_1\mathbf{A}_1,$$

где $\mathbf{A}_1 = (a_{11}, a_{21}, ..., a_{N1})^T$, $\mathbf{R}_1 = (r_{11}, 0, ..., 0)^T$. Это можно сделать, определив вектор $\mathbf{v}_1$ уравнением

$$\mathbf{A}_1 - 2\frac{\mathbf{v}_1^T\mathbf{A}_1}{\mathbf{v}_1^T\mathbf{v}_1}\mathbf{v}_1 = \mathbf{R}_1. \tag{C.2}$$

Векторное уравнение (С.2) эквивалентно системе N скалярных уравнений относительно $N+1$-ой неизвестной величины $\mathrm{v}_{11}, \mathrm{v}_{21}, ..., \mathrm{v}_{N1}, r_{11}$. Для однозначного решения системы к ней необходимо добавить дополнительной уравнение (условие). Возьмём его в виде

$$2\frac{\mathbf{v}_1^T\mathbf{A}_1}{\mathbf{v}_1^T\mathbf{v}_1} = 1. \tag{C.3}$$

С учётом (С.3) уравнение (С.2) даёт $\mathbf{v}_1 = \mathbf{A}_1 - \mathbf{R}_1$. Теперь, подставив этот вектор в (С.3), получим

$$r_{11} = \pm\sqrt{\mathbf{A}_1^T\mathbf{A}_1} = \pm|\mathbf{A}_1|.$$

Выберем здесь знак «плюс», обеспечив положительность для r_{11}. Таким образом, вектор $\mathbf{v}_1$, формирующий искомое преобразование Хаусхолдера, должен быть равен

$$\mathbf{v}_1 = \mathbf{A}_1 - |\mathbf{A}_1|\mathbf{e}_1, \tag{C.4}$$

где единичный вектор $\mathbf{e}_1 = (1,0,...,0)$.

Сформируем теперь матрицу Хаусхольдера, обращающую в нуль все компоненты вектора $\mathbf{A}_n = (a_{1n}, a_{2n}, ..., a_{Nn})^T$, кроме n первых:

$$\mathbf{R}_n = \mathbf{H}_n \mathbf{A}_n,$$

где $\mathbf{R}_n = (r_{1n}, r_{2n}, ..., r_{nn}, 0, ..., 0)^T$. Как и $\mathbf{v}_1$, вектор $\mathbf{v}_n$ определяется системой уравнений вида (C.2):

$$\mathbf{A}_n - 2\frac{\mathbf{v}_n^T \mathbf{A}_n}{\mathbf{v}_n^T \mathbf{v}_n}\mathbf{v}_n = \mathbf{R}_n, \tag{C.5}$$

к которой следует добавить n дополнительных условий. Одно из них по-прежнему –

$$2\frac{\mathbf{v}_n^T \mathbf{A}_n}{\mathbf{v}_n^T \mathbf{v}_n} = 1,$$

а в качестве других примем равенства нулю $\mathrm{v}_{1,n} = \mathrm{v}_{2,n} = ... = \mathrm{v}_{n-1,n} = 0$. Эти условия позволяют получить следующее решение системы уравнений (C.5) для компонент вектора $\mathbf{v}_n$ при $n = 2,3,...,N-1$:

$$\begin{aligned} &\mathrm{v}_{k,n} = 0, \ k = 1,2,...,n-1; \\ &\mathrm{v}_{k,n} = a_{k,n} - \delta_{k,n}\sqrt{A_{k,n}^2 + A_{k+1,n}^2 + ... + A_{N,n}^2}, \ \ k = n, n+1, ..., N. \end{aligned} \tag{C.6}$$

Нетрудно заметить, что матрицы (C.1), сформированные на основе векторов (C.4) и (C.6), обладают следующим свойством

$$\mathbf{R}_{n-1} = \mathbf{H}_n \mathbf{R}_{n-1}, \ n = 2,3,...,N-1.$$

Оно позволяет из матрицы $\mathbf{A} = (A_1, A_2, ..., A_N)$, проводя последовательность преобразований $\mathbf{H}_{N-1}\mathbf{H}_{N-2}...\mathbf{H}_1$, получить верхнюю треугольную матрицу $\mathbf{R} = (R_1, R_2, ..., R_N)$:

$$\mathbf{H}_{N-1}\mathbf{H}_{N-2}...\mathbf{H}_1\mathbf{A} = \mathbf{R}. \tag{C.7}$$

С учётом симметричности и ортогональности матриц $\mathbf{H}_n$ преобразование (C.7) можно записать как QR-разложение

$$\mathbf{A} = \mathbf{H}_1\mathbf{H}_2...\mathbf{H}_{N-1}\mathbf{R} = \mathbf{Q}\mathbf{R}. \tag{C.8}$$

Таким образом, алгоритм QR-разложения матрицы имеет вид

$$\mathbf{Q} = \mathbf{H}_1\mathbf{H}_2 \ldots \mathbf{H}_{N-1},\ \mathbf{R} = \mathbf{H}_{N-1}\mathbf{H}_{N-2} \ldots \mathbf{H}_1\mathbf{A}\,. \qquad \text{(C.9)}$$

Пример реализации алгоритма (C.9) даёт Mathcad-функция *QR(A)*.

```
QR(A) := | QR_разложение_матрицы_A ← %
         | N ← rows(A) − 1
         | I ← identity(N + 1)
         | v ← A^⟨0⟩ − |A^⟨0⟩|·I^⟨0⟩
         | H ← I − 2·v·v^T / (|v|)^2
         | A ← H·A
         | Q ← H
         | for n ∈ 1 .. N − 1
         |     x ← A^⟨n⟩
         |     for m ∈ 0 .. n − 1
         |       x_m ← 0
         |     v ← x − |x|·I^⟨n⟩
         |     H ← I − 2·v·v^T / (|v|)^2
         |     A ← H·A
         |     Q ← Q·H
         | augment(Q ,A)
```

Отметим, что для повышения вычислительной эффективности функции *QR(A)* можно, в частности, учесть, что при умножении на матрицу $\mathbf{H}_n$ *n-1*-ая верхняя строка умножаемой матрицы не меняется, и использовать это обстоятельство при программировании алгоритма.

Приложение D. Свойства собственных значений и векторов

Перечислим основные свойства собственных значений и собственных векторов, которые часто используются при расчётах. Для наглядности проиллюстрируем их вычислениями в пакете Mathcad.

1. Собственные значения симметричных матриц с действительными элементами (вещественных матриц) действительны. Это же свойство имеет место и для эрмитовых матриц с комплексными элементами.

$$A := \begin{pmatrix} 1 & 3 & 5 \\ 3 & 8 & 4 \\ 5 & 4 & 3 \end{pmatrix} \qquad \text{eigenvals}(A) = \begin{pmatrix} 2.538 \\ -3.104 \\ 12.567 \end{pmatrix}$$

$$Ac := \begin{pmatrix} 1 & 3 & 5+2j \\ 3 & 8 & 4 \\ 5-2j & 4 & 3 \end{pmatrix} \qquad \text{eigenvals}(Ac) = \begin{pmatrix} 12.684 \\ -3.574 \\ 2.89 \end{pmatrix}$$

2. Положительно определенные матрицы, т.е. матрицы, для которых при любом $\mathbf{x} \neq 0$ $(\mathbf{x}, \mathbf{Ax}) \geq 0$, имеют положительные собственные значения.

$$Ap = \begin{pmatrix} 3.5 & 4.7 & 3.2 \\ 4.7 & 8.9 & 5.9 \\ 3.2 & 5.9 & 5 \end{pmatrix} \quad n := 0..2 \quad y_n := \text{rnd}(2) \qquad y = \begin{pmatrix} 0.195 \\ 0.189 \\ 1.863 \end{pmatrix}$$

$$y^T \cdot Ap \cdot y = (24.626)$$

$$\text{eigenvals}(Ap)^T = (0.644 \quad 0.964 \quad 15.792)$$

3. Собственные векторы симметричной матрицы, принадлежащие различным собственным значениям ортогональны: $(\mathbf{x}_n, \mathbf{x}_m) = |\mathbf{x}_n|^2 \delta_{n,m}$. Для матриц, не являющихся симметричными, такие собственные векторы линейно независимы.

$$\lambda := \text{eigenvals}(A) \qquad \lambda^T = (2.538 \quad -3.104 \quad 12.567)$$

$$x_0 := \text{eigenvec}(A, \lambda_0) \qquad x_1 := \text{eigenvec}(A, \lambda_1) \qquad x_2 := \text{eigenvec}(A, \lambda_2)$$

$$n := 0..2 \qquad m := 0..2 \qquad c_{n,m} := \left(x_n^T \cdot x_m\right)_0 \qquad c = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

4. Вещественные симметричные матрицы с различными собственными значениями $\lambda_1, \lambda_2, ..., \lambda_N$ имеют полную ортонормированную систему собственных векторов и приводятся к диагональному виду

$$\mathbf{A} = \mathbf{X}^{-1} \mathbf{\Lambda} \mathbf{X},$$

где $\mathbf{\Lambda} = diag(\lambda_1, \lambda_2, ..., \lambda_N)$ – диагональная матрица, а **X** – ортогональная матрица, столбцами которой служат собственные векторы матрицы **A**.

Примечание: Более общая формулировка этого свойства состоит в том, что к диагональному виду приводится матрица с различными собственными значениями.

$$X := \text{eigenvecs}(A) \qquad \text{eigenvals}(A) = \begin{pmatrix} 2.538 \\ -3.104 \\ 12.567 \end{pmatrix} \qquad X^{T} \cdot X = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$X^{T} \cdot A \cdot X = \begin{pmatrix} 2.538 & -1.588 \times 10^{-15} & 0 \\ -1.542 \times 10^{-15} & -3.104 & 0 \\ 0 & 1.051 \times 10^{-15} & 12.567 \end{pmatrix}$$

5. Собственные значения матрицы **B**, связанной с матрицей **A** преобразованием подобия

$$\mathbf{B} = \mathbf{S}^{-1}\mathbf{A}\mathbf{S},$$

где **S** – невырожденная матрица, совпадают с собственными значениями матрицы **A**. При этом собственные векторы матриц **A** и **B** связаны соотношением $\mathbf{x_B} = \mathbf{S}^{-1}\mathbf{x_A}$.

$$B := S^{-1} \cdot A \cdot S \qquad \text{eigenvals}(A) = \begin{pmatrix} 2.538 \\ -3.104 \\ 12.567 \end{pmatrix} \qquad \text{eigenvals}(B) = \begin{pmatrix} -3.104 \\ 12.567 \\ 2.538 \end{pmatrix}$$

$$S = \begin{pmatrix} 3.5 & 4.7 & 3.2 \\ 4.7 & 8.9 & 5.9 \\ 3.2 & 5.9 & 5 \end{pmatrix} \qquad \text{eigenvecs}(B) = \begin{pmatrix} 0.765 & 0.42 & -0.488 \\ 0.025 & 0.738 & 0.674 \\ -0.643 & 0.528 & -0.555 \end{pmatrix}$$

$$X := \text{eigenvecs}(A) \qquad Z := S^{-1} \cdot X$$

$$\frac{Z^{\langle 0 \rangle}}{\left| Z^{\langle 0 \rangle} \right|} = \begin{pmatrix} 0.488 \\ -0.674 \\ 0.555 \end{pmatrix} \qquad \frac{Z^{\langle 1 \rangle}}{\left| Z^{\langle 1 \rangle} \right|} = \begin{pmatrix} 0.765 \\ 0.025 \\ -0.643 \end{pmatrix} \qquad \frac{Z^{\langle 2 \rangle}}{\left| Z^{\langle 2 \rangle} \right|} = \begin{pmatrix} -0.42 \\ -0.738 \\ -0.528 \end{pmatrix}$$

6. Сумма диагональных элементов матрицы **A** (след матрицы) равна сумме её собственных значений. Определитель матрицы равен произведению её собственных значений: $Det(\mathbf{A}) = \lambda_1 \lambda_2 ... \lambda_N$.

$$\sum_{n=0}^{2} A_{n,n} = 12 \qquad \sum_{n=0}^{2} \lambda_n = 12 \qquad |A| = -99 \qquad \prod_{n=0}^{2} \lambda_n = -99$$

7. Собственные значения треугольной матрицы совпадают с её диагональными элементами: $\lambda_1 = a_{11}, \lambda_2 = a_{22}, ..., \lambda_N = a_{NN}$.

$$At := \begin{pmatrix} 1 & 3 & 5 \\ 0 & 8 & 4 \\ 0 & 0 & 3 \end{pmatrix} \qquad \text{eigenvals}(At) = \begin{pmatrix} 1 \\ 8 \\ 3 \end{pmatrix} \qquad Y := \text{eigenvecs}(At)$$

$$Y^{-1} \cdot At \cdot Y = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

8. Собственные значения обратной матрицы обратны собственным значениям исходной матрицы: $\lambda\left(\mathbf{A}^{-1}\right) = \left[\lambda(\mathbf{A})\right]^{-1}$, а их собственные векторы совпадают.

$$\lambda inv := \text{eigenvals}\left(A^{-1}\right) \qquad \lambda inv^{T} = (\,0.08 \quad 0.394 \quad -0.322\,)$$

$$\lambda := \text{eigenvals}(A) \qquad \frac{1}{\lambda_0} = 0.394 \qquad \frac{1}{\lambda_1} = -0.322 \qquad \frac{1}{\lambda_2} = 0.08$$

$$\text{eigenvec}\left(A^{-1}, \lambda inv_0\right) = \begin{pmatrix} 0.42 \\ 0.738 \\ 0.528 \end{pmatrix} \qquad \text{eigenvec}\left(A, \lambda_2\right) = \begin{pmatrix} 0.42 \\ 0.738 \\ 0.528 \end{pmatrix}$$

$$\text{eigenvec}\left(A^{-1}, \lambda inv_1\right) = \begin{pmatrix} 0.488 \\ -0.674 \\ 0.555 \end{pmatrix} \qquad \text{eigenvec}\left(A, \lambda_0\right) = \begin{pmatrix} 0.488 \\ -0.674 \\ 0.555 \end{pmatrix}$$

$$\text{eigenvec}\left(A^{-1}, \lambda inv_2\right) = \begin{pmatrix} -0.765 \\ -0.025 \\ 0.643 \end{pmatrix} \qquad \text{eigenvec}\left(A, \lambda_1\right) = \begin{pmatrix} -0.765 \\ -0.025 \\ 0.643 \end{pmatrix}$$

9. Отношение Рэлея

$$\rho(\mathbf{u}) = \frac{(\mathbf{u}, \mathbf{A}\mathbf{u})}{(\mathbf{u}, \mathbf{u})}$$

для любой симметричной матрицы $\mathbf{A}$ и произвольного ненулевого вектора **u** находится в пределах

$$\lambda_{\min} \le \rho(\mathbf{u}) \le \lambda_{\max}.$$

Если $\mathbf{x}_n$ – собственный вектор, то $\rho(\mathbf{x}_n) = \lambda_n$.

$$\rho(u) := \frac{\left(u^T \cdot A \cdot u\right)_0}{u^T \cdot u} \qquad \text{eigenvals}(A) = \begin{pmatrix} 2.538 \\ -3.104 \\ 12.567 \end{pmatrix}$$

$$n := 0..2 \qquad u_n := \text{rnd}(2) \qquad \rho(u) = (9.288)$$

$$\rho(\text{eigenvec}(A, 2.538)) = (2.538)$$

$$\rho(\text{eigenvec}(A, -3.104)) = (-3.104)$$

$$\rho(\text{eigenvec}(A, 12.567)) = (12.567)$$

Приложение E. Итерации на основе отношения Рэлея

Отношение Рэлея имеет следующее свойство: если $\mathbf{x} = \mathbf{x_n} + \xi$, причём $\|\xi\| / \|\mathbf{x}_n\| \sim \mu << 1$, то

$$\frac{|\rho(\mathbf{x}) - \lambda_n|}{|\rho(\mathbf{x})|} \sim \mu^2.$$

Это позволяет использовать отношение в процессе последовательных приближений к собственному вектору $\mathbf{x_n}$ и собственному значению λ_n вида:

$$\left(\mathbf{A} - \rho(\mathbf{w}^{(k)})\mathbf{I}\right)\mathbf{w}^{(k)} = \mathbf{w}^{(k-1)}. \qquad \text{(E.1)}$$

Алгоритм (E.1) реализуется Mathcad-функцией *EGR(A,λ,ε)*. Пример вычислений – программа (Pr.E).

$$A = \begin{pmatrix} 1 & 3 & 5 \\ 3 & 8 & 4 \\ 5 & 4 & 3 \end{pmatrix} \qquad \text{eigenvals}(A) = \begin{pmatrix} 2.537665 \\ -3.104404 \\ 12.566739 \end{pmatrix}$$

$$\text{EGR}(A, 1, 0.001)_0 = 2.537665 \qquad \text{EGR}(A, -1, 0.001)_0 = -3.104404$$

$$\text{EGR}(A, 7, 0.001)_0 = 12.566739 \qquad \text{EGR}(A, 7, 0.001)_4 = 4$$

(Pr.E)

$$
\begin{aligned}
&\mathrm{EGR}(A,\lambda 0,\varepsilon) := \\
&\quad \text{Итерации\_с\_отношением\_Рэлея} \leftarrow \% \\
&\quad \text{для\_произвольного\_СЗ\_матрицы} \leftarrow \% \\
&\quad N \leftarrow \mathrm{rows}(A) \\
&\quad I \leftarrow \mathrm{identity}(N) \\
&\quad w \leftarrow A^{\langle 0 \rangle} \\
&\quad B \leftarrow A - \lambda 0 \cdot I \\
&\quad w \leftarrow \mathrm{lsolve}(B,w) \\
&\quad \rho_0 \leftarrow \frac{\left(w^T \cdot B \cdot w\right)_0}{\left(w^T \cdot w\right)_0} \\
&\quad k \leftarrow 1 \\
&\quad \text{while } 1 \\
&\qquad \rho_k \leftarrow \frac{\left(w^T \cdot A \cdot w\right)_0}{\left(w^T \cdot w\right)_0} \\
&\qquad w \leftarrow \mathrm{lsolve}\left(A - \rho_k \cdot I, w\right) \\
&\qquad w \leftarrow \frac{w}{|w|} \\
&\qquad \text{return } \mathrm{stack}\left(\rho_k, w, k\right) \text{ if } \left|\rho_k - \rho_{k-1}\right| \le \varepsilon \\
&\qquad k \leftarrow k + 1
\end{aligned}
$$

В *EGR(A,λ,ε)* первое приближение к собственному вектору вычисляется с помощью обратного степенного метода с фиксированным сдвигом.

Библиографический список

1. *Бахвалов Н.С., Жидков Н.П., Кобельков Г.М.* Численные методы. – 3-е изд., доп. и перераб. М.: БИНОМ. Лаборатория знаний, 2011. 636 с.
2. *Косарев В.И.* 12 лекций по вычислительной математике. – 3-е изд., испр. и доп. М.: Физматкнига, 2013. 240 с.
3. *Зализняк В.Е.* Численные методы. Основы научных вычислений. – 2-е изд., перераб. и доп. М.: Юрайт, 2012. 356 с.
4. *Костомаров Д.П., Фаворский А.П.* Вводные лекции по численным методам. М.: Логос, 2004. 184 с.
5. *Горбаченко В.В.* Вычислительная линейная алгебра с примерами на MATLAB. СПб: БХВ-Петербург, 2011. 320 с.
6. *Уоткинс Д.* Основы матричных вычислений. М.: БИНОМ. Лаборатория знаний, 2006. 664 с.
7. *Мэтьюз Д., Финк К.* Численные методы. Использование MATLAB. М.: Издательский дом «Вильямс», 2001. 720 с.
8. *Голуб Дж., Ван Лоун Ч.* Матричные вычисления. М.: Мир, 1999. 548 с.
9. Численные методы. Сборник задач / Под ред. У.Г. Пирумова. М.: Дрофа, 2007. 144 с.

СОДЕРЖАНИЕ

Учебное издание

Зайцев Валерий Васильевич

ЧИСЛЕННЫЕ МЕТОДЫ ДЛЯ ФИЗИКОВ.
СИСТЕМЫ ЛИНЕЙНЫХ УРАВНЕНИЙ
И МАТРИЧНЫЕ ВЫЧИСЛЕНИЯ

Учебное пособие

В авторской редакции

Подписано в печать 20.12.2017. Формат 60x84 1/16.
Бумага офсетная. Печ. л. 4,75.
Тираж 25 экз. Заказ .

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ имени академика С.П. КОРОЛЕВА»
(Самарский университет)
443086, Самара, Московское шоссе, 34.

Изд-во Самарского университета.
443086, Самара, Московское шоссе, 34.